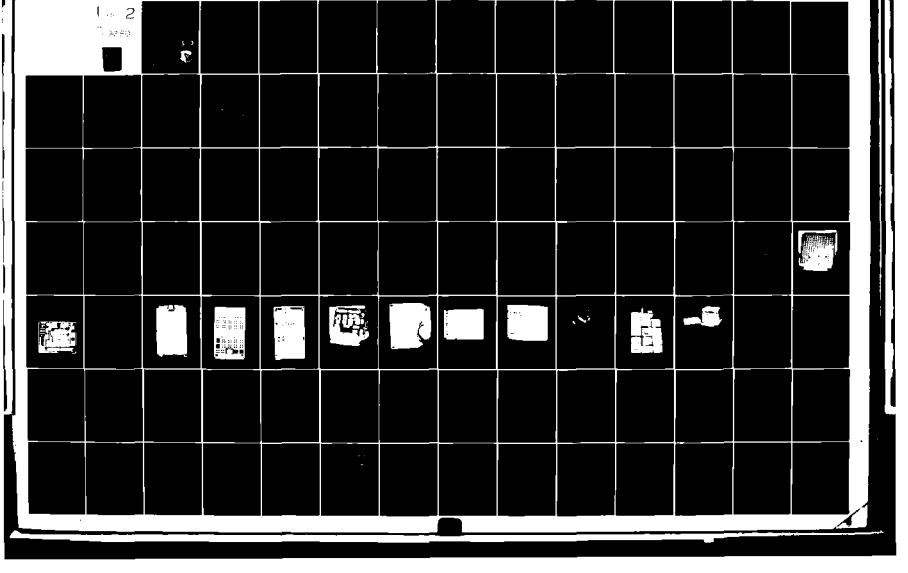
AD-A119 380 GENERAL DYNAMICS SAN DIEGO CA CONVAIR DIV F/G 9/2
DIGITAL INTEGRATING SUBSYSTEM (DIS).(U)
OCT 81 K D ARNOLD, D L BURGESS, J F FASSEL F08635-79-C-0206
UNCLASSIFIED CS64-32228 AFATL-TR-81-83 NL



AFATL-TR-81-83

Digital Integrating Subsystem (DIS)

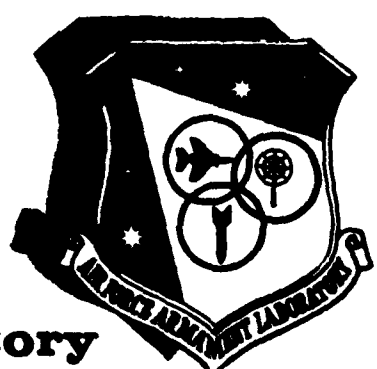
L D Arnold
J L Burgess
J F Fassel, et al

GENERAL DYNAMICS CONVAIR DIVISION
P O BOX 80847
SAN DIEGO, CALIFORNIA 92138

OCTOBER 1981

FINAL REPORT FOR PERIOD MAY 1979-MAY 1981

Approved for public release; distribution unlimited



Air Force Armament Laboratory
AIR FORCE SYSTEMS COMMAND★UNITED STATES AIR FORCE★EGLIN AIR FORCE BASE, FLORIDA

NOTICE

Please do not request copies of this report from the Air Force Armament Laboratory. Additional copies may be purchased from:

National Technical Information Service
5285 Port Royal Road
Springfield, Virginia 22161

Federal Government agencies and their contractors registered with Defense Technical Information Center should direct requests for copies of this report to:

Defense Technical Information Center
Cameron Station
Alexandria, Virginia 22314

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFATL-TR-81-83	2. GOVT ACCESSION NO. AD-A119 380	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) DIGITAL INTEGRATING SUBSYSTEM (DIS)		5. TYPE OF REPORT & PERIOD COVERED Final Technical Report 1 May 1979 - 15 May 1981
		6. PERFORMING ORG. REPORT NUMBER CS64-32228
7. AUTHOR(s) K.D.Arnold, D.L.Burgess, J.F.Fassel, E.F. Godenschwager, B.J. Haire, M.J. Koenekamp, H.G.Mayer, K.E. Saunders, W.G. Schneider, R.E. Wisniewski		8. CONTRACT OR GRANT NUMBER(s) F08635-79-C-0206
9. PERFORMING ORGANIZATION NAME AND ADDRESS General Dynamics Convair Division P.O. Box 80847 San Diego, CA 92138		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS Program Element: 63601F JDN: 670B-02-34
11. CONTROLLING OFFICE NAME AND ADDRESS Air Force Armament Laboratory Armament Division Eglin AFB, FL 32542		12. REPORT DATE October 1981
		13. NUMBER OF PAGES 168
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

Availability of this report is specified on verso of front cover.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

cross assembler, diagnostic software, digital processing, higher order language, JOVIAL (J73) compiler, microprocessor, MIL-STD-1553B, multiplex data bus, operating system, real-time computing, Zilog Z8000.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The DIS program has been successful in demonstrating that a small, potentially low-cost, high order language, airborne computer is possible. Twenty-five of these units have been delivered after extensive testing of both software and hardware compatibility. In addition, DIS Diagnostic Stations, used to host source and object JOVIAL files as well as all necessary programming tools, were also designed, tested, and delivered.

DD FORM 1473 EDITION OF 1 NOV 65 IS OBSOLETE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Block 20 (Continued)
The use of a distributed microprocessor system will provide a number of benefits yet to be verified in such programs as Midcourse Guidance Demonstration (MGD), Medium Range Air to Surface Missile (MRASM), and low-cost tactical systems. Integration of various subsystems is facilitated by the use of the DIS distributed, modular architecture.

However, even in light of the positive aspects of the DIS systems, a number of concerns have surfaced and must be addressed if such a system is actually projected for use in a full scale engineering development (FSED) weapon system. These concerns either have met with little success (mature hardware and software in a cost-effective production form) or no success (establishment of interface standards and specifications).

PREFACE

This report is the final technical report written to satisfy the contractural obligation of the Digital Integrating Subsystem (DIS), Contract F08635-79-C-0206 with the Air Force Armament Laboratory as the sponsor, Capt Rich Butler AFATL/DLMM program manager.

It was prepared by General Dynamics Convair Division, P. O. Box 80847, San Diego, California 92138 for the Armament Division DLMM, Eglin AFB, Florida 32542, the sponsor. Softech, Inc., San Diego, was retained as a subcontractor for software support concerning the compiler.

The Public Affairs Office has reviewed this report, and it is releasable to the National Technical Information Service, where it will be available to the general public, including foreign nationals.

This technical report has been reviewed and is approved for publication.

FOR THE COMMANDER

Hubert L Redmon

HUBERT L. REDMON, Lt Colonel, USAF
Acting Chief, Guided Weapons Division

(The reverse of this page is blank)

SUMMARY

The Digital Integrating Subsystem (DIS) Program is an advanced technology computer development program for the Air Force, which uses state-of-the-art components and has a unit design-to-cost objective of less than ten thousand dollars ($10K) (constant fiscal year (FY) 78 dollars for a quantity of 1000 units). The DIS Program was to provide the capability to tailor a computer for a mission application by the substitution of various input/output circuit boards in a modular system that was small and flight-rated. Five shipsets, each containing five DIS computers, have been delivered to the Air Force.

Some of the technical requirements imposed on the DIS design were:

- Hardware Characteristics
  - Class I central processing unit (CPU) $\geq$ 300 thousand operations per second (KOPS)
  - Class II CPU $\geq$ 500 KOPS
  - Low Power $\leq$ 50 watts in specified configuration
  - Low Volume $\leq$ 150 in^3
  - Low Weight $\leq$ 6 lb
  - Multiple Input/Output capability
  - Memory up to 32K words of random access memory (RAM) and up to 32K words of read only memory (ROM)
  - Meet the specified environment (normal cruise missile flight environment)
- Input/Output Characteristics
  - Direct Memory Access (DMA): 500K words/sec, bidirectional, low-power Shottky transistor-transistor logic (LSTTL) interface.
  - Parallel: Peripheral initiated input, computer initiated output 250K words/sec, LSTTL interface.
  - Serial: Peripheral initiated input, computer initiated output, computer supplied clock of 110, 300, 1200, 9600, 250K, and 500K bits/sec, LSTTL interface.
  - 1553B: Fully MIL-STD-1553B compatible remote terminal, dual-channel implementation using two cards.

- Bus Interface Unit (BIU): Internal missile bus, simplex channel, round-robin protocol.

- DIS Diagnostic Station (DDS) Characteristics

  - Power, monitor and control up to eight DIS computers via BIU.

  - Host DIS object code, linker/loader and control, monitor and load (CML).

  - Provide user interfaces cathode ray tube (CRT), printer, Advanced Research Project Agency Compiler Network (ARPANET).

  - Provide JOVIAL link between host and DIS computer.

  - Diagnose DIS/DDS input/output (I/O) card failures where possible.

- Software Characteristics

  - Retarget 370/370 Jovial Compiler and Implementation Tool (JOCIT) JOVIAL J73 compiler to DIS (Z8002).

  - Develop for DIS an assembler, linker/loader, editor, CML, a real-time operating system, diagnostics (for DDS also), and acceptance test software.

  - Host the compiler on the IBM 370.

  - Write airborne software in J73.

The hardware technical requirements listed above were met by the DIS design that was released to the experimental factory for fabrication.

A qualification test was completed including electromagnetic interference (EMI). All of the physical environmental testing was done at General Dynamics Pomona Division. There is one minor outstanding issue— the memory backup battery is leaking under high-humidity conditions. The battery vendor has changed the design of the header seal to alleviate this problem.

The DIS program had the normal problems typically associated with technology programs: late parts delivery, vendor preliminary specification discrepancies, and inherent design flaws in vendor's initial product release. An example of vendor design flaw was the Z8002 microprocessor dual inline package (DIP); the vendor had to make several step (block) changes to achieve an acceptable product for the DIS application.

The first shipset delivery was about three months later than the initial program schedule milestones. Most of this schedule slip was directly attributable to late delivery and configuration changes from preliminary specification data sheets of

state-of-the-art devices. The remaining schedule slip was due to incorporating design changes determined necessary during DIS computer checkout on the General Dynamics (GD) facility Federated Diagnostic Station (FDS). This was the first test of the computer's high speed I/O capability and some rate changes were required. The last shipset was delivered about two months late. The schedule improvement was due to a printed wiring board redesign, done at GD expense, that removed the hardwire changes necessitated by the design improvements. Three diagnostic stations were delivered during the execution of this contract. They currently support Air Force activities and Midcourse Guidance Demonstration (MGD) integration activities.

The delivered compiler was based on the B5 JOCIT front end. This version successfully executed its entire possible test suites allowed by the 370 front end. Because of the remaining compiler deficiences, the B5 version was upgraded to the B7 version. During this new debugging process, the characteristics of B7 were enhanced to B8. Compiler maintenance after this upgrade is now the responsibility of the program involved.

All other software packages, most of which are written in FORTRAN, were successfully written, executed, and acceptance tested.

TABLE OF CONTENTS

TABLE OF CONTENTS (CONTINUED)

TABLE OF CONTENTS (CONCLUDED)

LIST OF FIGURES

LIST OF FIGURES (CONCLUDED)

LIST OF TABLES

(The reverse of this page is blank)

SECTION I

DIS COMPILER

INTRODUCTION

In 1979 General Dynamics (GD) was required to produce a JOVIAL J73 compiler with two code generators. The input or source language was JOVIAL J73, as defined in MIL-STD 1589A by the USAF; the two planned targets were the PDP-11 minicomputer and Z8002 microprocessor. Initial scheduled delivery was mid-1979.

The short time frame was not expected to impose any problem, since Softech's already existing front-end of the JOCIT compiler, supposedly written in a machine independent way, was to be used for both code generators.

Ultimately, the same company, Softech, was put in charge of designing and developing the compiler. A GD employee was expected to participate in the development process to allow a painless and complete transfer of knowledge from Softech to GD.

OBJECTIVE

The same philosophy, to keep the front-end target independent, was applied to the code generator to the extent possible. This resulted in the following architecture: all code sequences were kept target independent; for each code generator (in the DIS case only the one for Z8002) a database was provided, to be referenced by the code. Thus the addition of further code generators essentially requires the design of a new database, and only minor code modifications.

ARCHITECTURE

This paragraph gives a rough outline of the DIS compiler's code generator two-pass architecture; for more information consult the DIS J73 JOVIAL "Computer Program Design Document," Vol I.

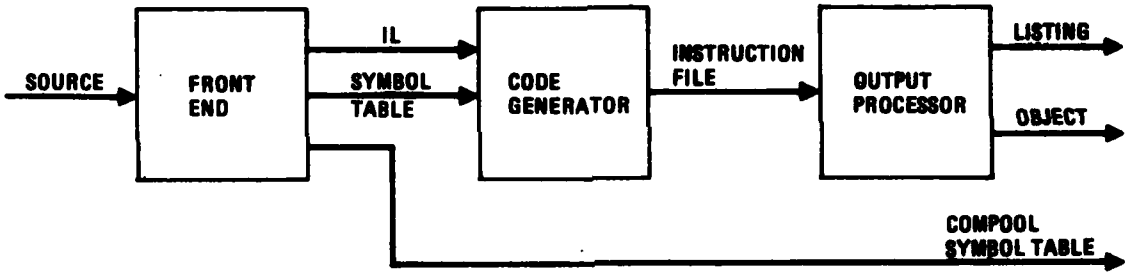
The DIS code generator is a complete replacement of the IBM code generator. The interface between the front-end of the compiler and the code generator (CG), the symbol table, and the intermediate language (IL), remain unchanged. The interface between the code generator and the output formatter and the instruction file has been modified to provide for the special requirements of the code generator design as well as to better represent the instructions on the Z8002 processor.

The DIS code generator performs two passes over the source program. The first pass, performed by CG1, involves processing the program as it is repre-

sented in the IL and the symbol table. The second pass, performed by CG2, involves processing the program as it is represented in the instruction file generated by CG1 and the symbol table. CG2 transforms the instruction file into a format that can be processed by the output processor.

The task of CG1 is to translate every executable statement in the source program into a semantically equivalent Z8002 code sequence. The task of CG2 is to perform forward label resolution within the instruction generated, macro expansions, and (optionally) peephole optimization of generated code, and to generate variable, constant, and absolute data program segments. CG1 and CG2 share some common procedures to interface with the instruction file and to determine the size of instructions as they are emitted into the instruction file.

Below is a schematic picture of the code generator, with the whole front-end simply represented by one box.



DIS Compiler Architecture

SECTION II

DIS CROSS ASSEMBLER

INTRODUCTION

The DIS Cross Assembler (DCA) is written as a two-pass, symbolic assembler consistent with widely accepted criteria for such an assembler. It produces object code to be processed by a relocating loader from source code written symbolically by a programmer or produced as the output of a compiler. The DCA was required because no other cross assembler for the Z8002 processor was commercially available that executed on the PDP 11/34 and met the requirements of the DIS computer. In addition, writing the DCA internally provides the option to tailor the assembler to specific needs, e.g., the creation of floating point numbers in the DIS format.

OBJECTIVE

The objective of the DCA was to provide the capability of translating symbolic source code into relocatable object code to aid in debugging the JOVIAL J73 compiler output and to allow assembly language coding of some DIS systems programs. Since the JOVIAL J73 compiler was not completed in time for the programming of the DIS Operating System (OS), the DCA was used, and symbolic assembly language was selected as the operating system source language. The DCA uses the same mnemonics as the Zilog PLZ/ASM assembler for machine opcode and has a set of pseudo opcodes developed to fit the requirements of the DIS program. At the time the DCA was developed the Zilog pseudo opcodes had not been defined. The DCA, in addition to producing the object code, produces a listing of the translated source code, a symbol cross reference, and a memory segmentation map. The cross reference and segmentation map serve as debug aides to the user.

APPROACH

The functional requirements of the DCA were defined in specification CS64-32201, Revision A, dated 24 September 1980 (the Part 1 specification of the DIS Cross Assembler). The input source language for the DCA was defined by the mnemonics of the Zilog PLZ/ASM assembler and a set of pseudo opcodes chosen to satisfy DIS requirements. The names of symbols are restricted to six or fewer characters to be consistent with the Digital Equipment Corporation (DEC) librarian program. The output object code of the DCA follows the DEC relocatable object code format; this also was done for compatibility with the DEC librarian program. The processing of source code by the DCA follows traditional concepts. During pass I a symbol table is built to allow forward references to symbols to be resolved during pass II. During pass II object code is developed using the symbol table for label references and using the reserved symbol table for mnemonic references. Because the size of an instruction depends

on the addressing mode used, the address field of each instruction must be partially processed during pass I; this would not have to be done for a computer with a fixed length for each instruction. The hashing access method is the same for both the symbol table and the reserved symbol table. The hashing technique selected emphasizes speed of retrieval, flexibility of table length (note that symbols are limited to six or fewer characters), and coding ease. Modules of the DCA were grouped to form overlay structures since the total size of the assembler exceeded machine capacity of 32K words for an executable program. The groupings of modules into overlays is ordered so that the swapping of overlays is minimized. The DCA was coded in FORTRAN IV since the JOVIAL J73 compiler was not available in time.

RESULTS

The DCA was tested in three stages: module level testing, integration testing, and acceptance testing. The DCA was developed in a top down manner, which influences the testing procedure. The control modules are programmed first and then module tested. Stubs for the remaining modules to be developed are then added to the control modules. This grouping of control modules and stubs provides the environment into which all subsequent modules are integration tested. Before entering a module for integration test it was module tested to ensure that all program paths were traversed and that the module interface was as specified in the design. When a module was entered into the integration test environment its interface to the previously integrated modules was tested. When all the module stubs were filled out, the complete assembler was used to generate object code to run on a DIS computer. Finally, a formal acceptance test was run to test every instruction and every addressing mode of each instruction; also, each pseudo opcode was tested. The resulting object code was compared against a list of expected results. The formal acceptance test is described in document number CS64-30102A dated 1 September 1980.

CONCLUSIONS/RECOMMENDATIONS

After extensive use of the DCA in developing the DIS OS, several conclusions were reached regarding the efficiency and the user interface of the DCA. They are:

a. The assembler would execute faster and use less memory if it were coded in assembly language rather than FORTRAN IV.

b. The assembler would execute faster if it were configured to have two sizes of symbol tables. A small symbol table of approximately 500 entries would not require the use of a virtual memory array and would satisfy most user requirements. A large symbol table (5000 entries) would be available when required.

c. The use of structured coding techniques requires more memory and reduces execution speed.

d. Additional features that should be added:

1. Nested macro capability – this is a useful tool to standardize certain code sequences and to speed up program development.
2. Conditional assembly – this feature would be useful especially for maintaining different configurations of programs such as the DIS OS.
3. Fixed point constants – a new pseudo opcode would create fixed point constants as is now done for decimal, hexadecimal, and floating point constants.

SECTION III

DIS DIAGNOSTIC STATION (DDS)
LINKING LOADER SOFTWARE

INTRODUCTION

The potentially complex structure of an executable program targeted for a Digital Integrated Subsystem (DIS) computer necessitates a method by which the individual structural components can be linked together and assigned memory locations with a high degree of flexibility. The DIS Linking Loader (DLL) program is the means by which these requirements are met.

The linking loader program is executed on the DIS Diagnostic Station (DDS) host computer, the Digital Equipment Corporation (DEC) PDP-11/34. Initiated from and operated under the DEC RSX-11M real-time operating system, the linking loader program effectively and efficiently loads and links DIS relocatable object code modules into a single, absolute DIS executable program. Using the DDS control, monitor, and load (CML) function, this program can then be down-loaded into either class DIS computer, to execute under the control of the DIS OS.

This section presents various aspects of the DLL design philosophy, concluding with suggestions and recommendations for program enhancement.

OBJECTIVES

The main purpose of the DLL program is to create an executable program capable of being down-loaded into either class DIS computer. The components of the executable program are relocatable object code modules produced by the JOVIAL J73 Compiler (hosted on the IBM 370) and the DCA (hosted on the PDP-11/34). These modules are stored on object code files on the DDS disk cartridge mass storage system and are capable of being combined into library files using the DEC Librarian as described in the IAS/RSX-11 Utilities Procedures Manual.

Using a video terminal (CRT) or Decwriter terminal to enter load directives, modules from both object code files and library files can be loaded (i.e., assigned memory locations) and linked by the linking loader program to produce an executable program file. These modules are: 1) the DIS OS modules, 2) user-defined application tasks, and 3) specified outside application task boundaries, i.e., global. The modules are specified (either directly or indirectly) in the input load directive sequence.

A summary of the processing capabilities of the DLL can be found in Paragraph 3.3 of the Linking Loader User's Guide (CS64-32205).

APPROACH

Function Allocation

The DLL program is a computer program configuration item (CPCI) whose functions are divided between six computer program components (CPCs). The allocation of these functions is listed in Table 1.

TABLE 1. DLL CPC FUNCTION ALLOCATION

CPC	Function
DLLCTL	Has overall DLL control responsibility.
DLLDRP	Accept user inputs defining boundaries of various regions of DIS memory. Accept user inputs defining DIS application tasks. Detect and list diagnostic messages for directive processing.
DLLGSD	Maintain separate load counters for instructions, fixed and variable data. Provide for searching of libraries to satisfy externally referenced modules. Generate and preserve a load symbol table for symbolic debugging. Detect and list diagnostic messages for load processing.
DLLTRP	Perform relocation of independently assembled and compiled DIS programs. Detect and list diagnostic messages for load processing.
DLLXPG	Generate absolute core image checksums. Retrieve all data necessary for generating an executable core image for the DIS computers. Generate a memory load map describing the allocation of storage, the modules that make up the load image, and the value of all global symbols. Detect and list diagnostic messages for load processing.
DLLCON	Generate a concordance list of all external symbols and global data blocks.

Disk Overlay Structure

The size of the DLL program necessitated the design of a disk-resident overlay scheme. Figure 1 depicts the resulting CPCI structure.

Each name in Figure 1 represents a program segment brought into core from the disk as it is needed. The "root" segment is DLLCTL and is always in core. DLLCTL is the control CPC and calls the remaining five CPCs in succession. In Figure 1, the leftmost six segment names are also the names of the six CPCs.

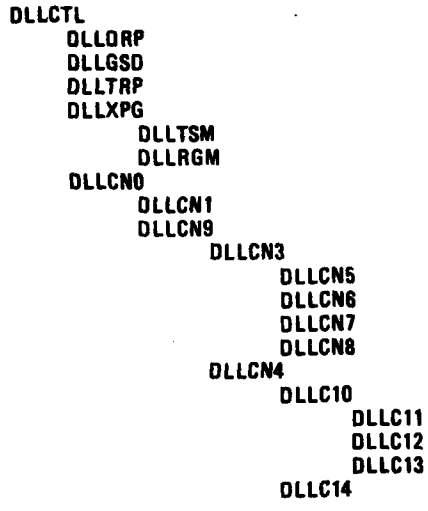


Figure 1. DLL Disk Overlay Structure

The first four CPCs each contain only the similarly named segment. The DLLXPG CPC is composed of the segments DLLXPG, DLLTSM, and DLLRGM. The DLLCON CPC contains 14 segments – DLLCON and the 13 others under it in the diagram.

Figure 1 shows the segments that are already in core when another segment is first read from the disk. Notice the segments are arranged in six "levels" from left to right. When a given segment is brought into core, the segment immediately above and to the left is already in core (and so on, progressing a level at a time from right to left). For example, when segment DLLC10 is read from the disk, segments DLLCN4, DLLCN9, DLLCON, and DLLCTL are already in core.

Functional Flow

A top-level chart depicting the functional flow of the DLL is provided in Figure 2. The control CPC (DLLCTL) functions as the interface between the RSX-11M operating system (OS) and the DLL. The control CPC calls the main processing routines of the remaining five CPCs. This processing is halted if an ABORT load directive is entered or if any kind of fatal error is detected. The latter results in a diagnostic being issued. Processing for the control CPC ends with the collected lineprinter file and OUTPUT file contents (if any) being printed and saved, respectively, and all remaining opened files being closed. Processing control is then returned to the RSX-11M OS.

File Descriptions

A description of the various disk files generated and used by the linking loader can be found in paragraphs 3.7.1.1.1 through 3.7.1.1.13 of the Linking Loader

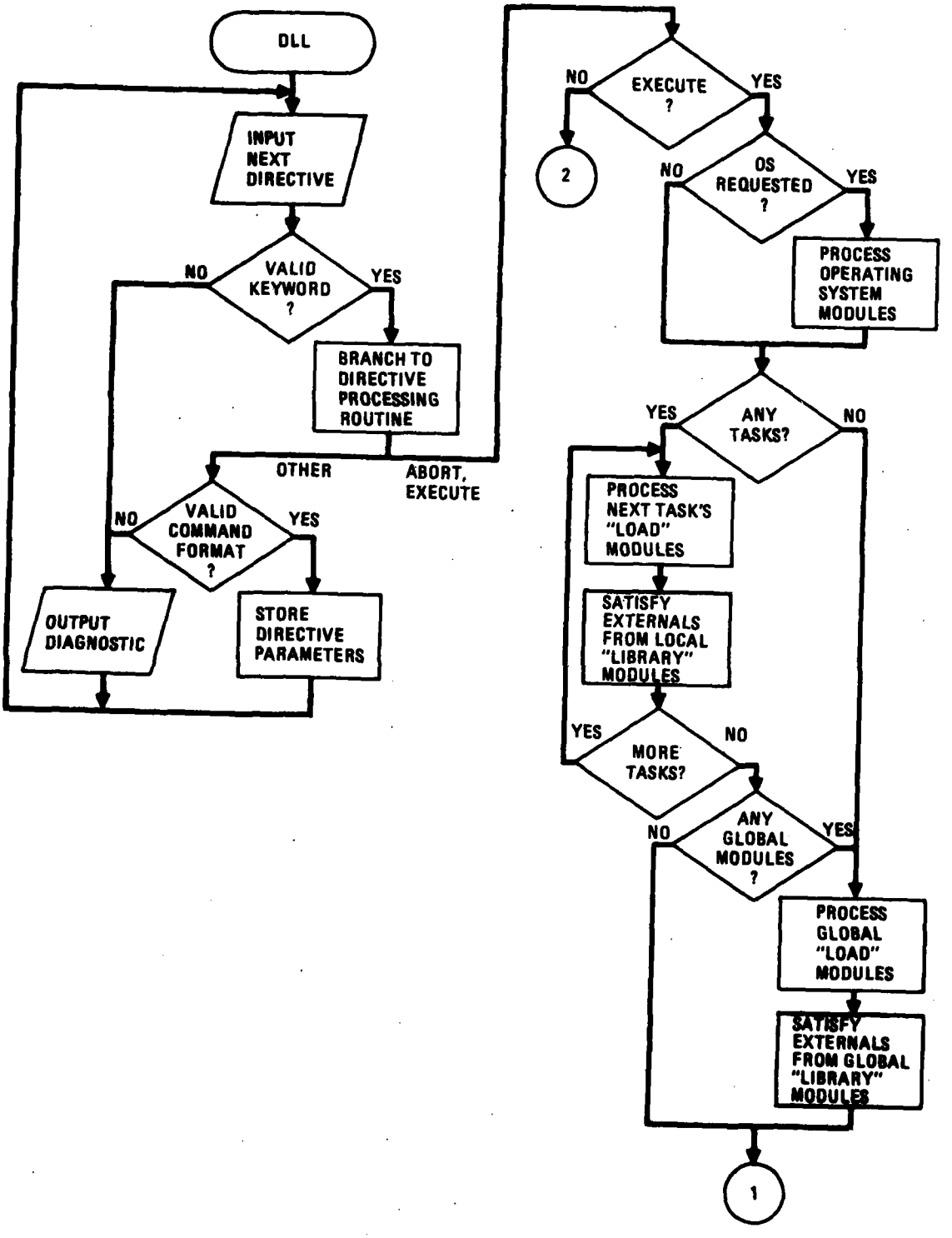


Figure 2. DLL Functional Flow

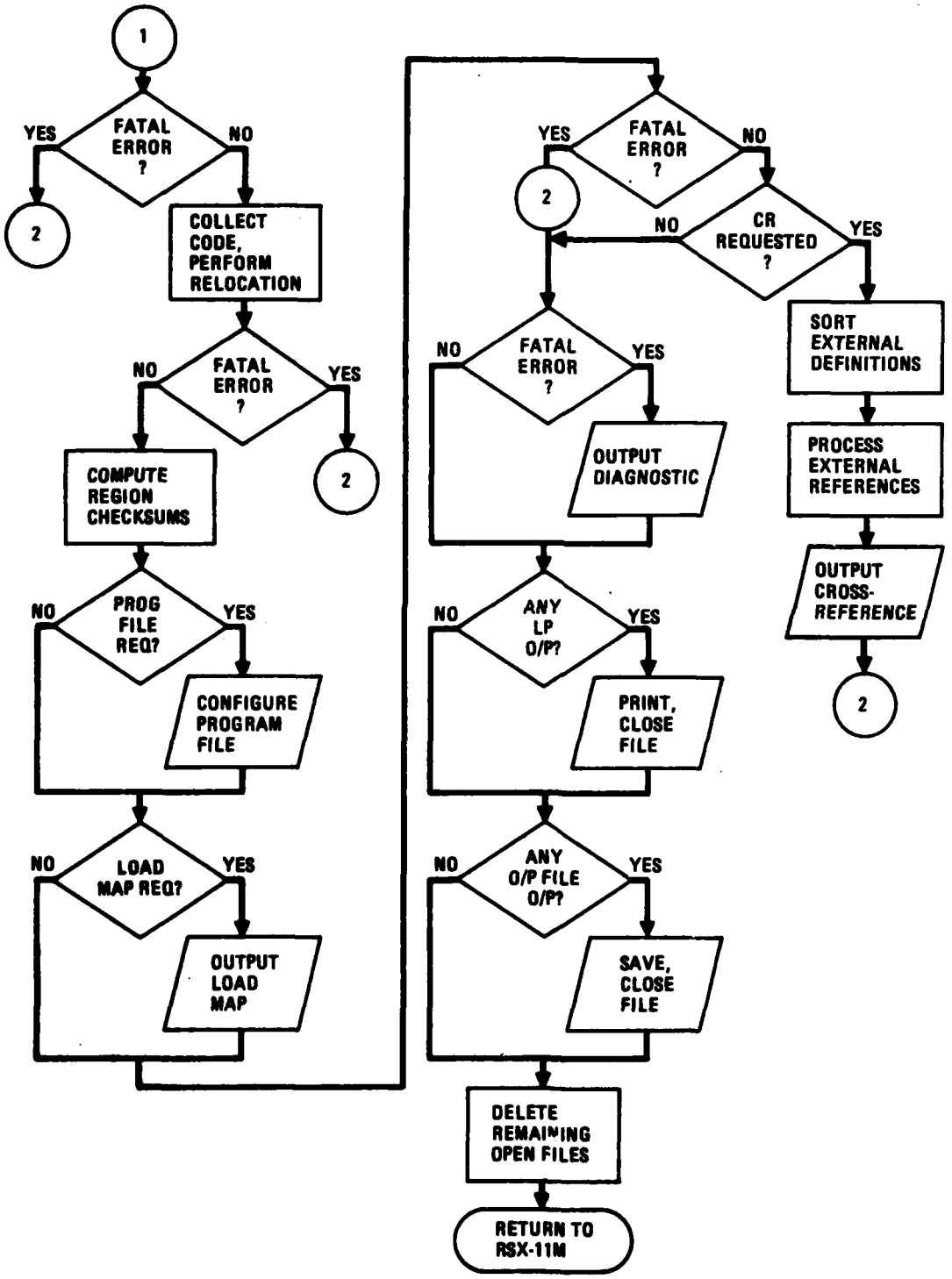


Figure 2. DLL Functional Flow (Concluded)

Part II Specification (CS64-32202). The most complete description of the executable program file is found in paragraph 5.3.3 of the Linking Loader User's Guide (CS64-32205).

Load Directive Usage

Complete explanations concerning the use of the DLL load directives are presented in paragraphs 6.2 through 6.2.17 of the Linking Loader User's Guide (CS64-32205). A summary of the directives and their functions is provided in Table 2.

TABLE 2. DIS LINKING LOADER DIRECTIVES

Command	Abbreviation	Options	Description
ABORT	A		Abort DLL, return to operating system.
CROSS-REFERENCE	CR	{ON | OFF}	External cross-reference generation switch. Default value is OFF.
DIRECTIVES	DI	{ON | OFF}	Directive echo switch. Default value is OFF.
DISPLAY	D		Display file and module names and options for current load sequence.
END-TASK	ET		Delimiter for task definition directives.
EXECUTE	X		Perform load processing, return to operating system.
IMAGE	IM	filename	Specifies executable program file name.
INPUT	IN	{filename | CRT}	Specifies load directive input file name or device. Default value is CRT.
LIBRARY	LB	filename [,...]	Specifies library files from which externals are to be satisfied.

TABLE 2. DIS LINKING LOADER DIRECTIVES (CONTINUED)

Command	Abbreviation	Options	Description
LOAD	LD	filename [(modulename[,...])] [,...]	Specifies files from which object code modules are to be loaded.
MAILBOX	MB	{number 1-63} = {DMA / 1553 / BIU / NULL} [,...]	Specifies mailbox numbers and their respective types.
MAP	M	{ON / OFF}	Load map generation switch. Default value is ON.
OPERATING-SYSTEM	OS	{NONE / ALL}	Specifies operating system configuration. Default value is ALL.
		{FS / TS / FIXEDSTORAGE / TASKSTORAGE} {length / length. / length}	Specifies length of OS Fixed Storage Area (default value is 460 (10) words) or number of words per task in the OS Storage Area (default value is 54 (10) words).
OUTPUT	O	{LP / LINE-PRINTER / CRT / filename} [,...]	Assigns list output device(s) and/or file. Default value the line-printer.
REGION	RG	{D / I / DATA / INSTRUCTIONS} ({'fwa / fwa. / fwa}, {lwa / lwa. / lwa})	Specifies core region boundaries between which all data and instructions are to be loaded.

TABLE 2. DIS LINKING LOADER DIRECTIVES (CONCLUDED)

Command	Abbreviation	Options	Description
SET	S	{V, C, I, A, VARIABLES, CONSTANTS, INSTRUCTIONS, AUXILIARY} = {'fwa, fwa., fwa} [,...]	Set location counter for program region.
TASK	T	{number} 1-32767 , {priority} 1-255, {A, D, ACTIVE, DORMANT} , [{' length, length., length} 0-7FFF]	Specify task attributes, begin task definition.

Summary of DIS Program Generation Processing

DIS OS modules, unless not required at all, are loaded and linked first. OS modules cannot reference, nor be referenced from modules outside the operating system.

Task definitions (if present in the load sequence) are processed next in their input order. Each task is loaded and linked independently of other tasks.

LOAD files are processed in their input order within a task. Resulting unsatisfied externals are searched for in the task's LIBRARY files. Only referenced modules from these files are loaded. Any externals not satisfied locally are considered global.

Next, the global LOAD files are processed in their input order. Afterwards, any unsatisfied global externals are searched for in global LIBRARY files. Only referenced modules from these files are loaded.

A LIBRARY file set (i.e., LIBRARY files within a specific task, or all global LIBRARY files) is searched in load directive input order. The first library is searched for modules needed to satisfy all externals that can be satisfied from the library. This process is repeated for each library in turn. After the last library is searched, processing returns to the first library if all externals are not yet satisfied. This circular search continues until all externals are

satisfied or it is established that no more externals can be satisfied from any of the libraries.

Program Image

Figure 3 depicts a typical load pattern within a DIS computer. Unless a REGION directive is entered (causing the entire program to be loaded in one memory half or the other), variables and constants are loaded in the lower (data) half of memory and the instructions and auxiliary regions are loaded in the upper (instructions) half.

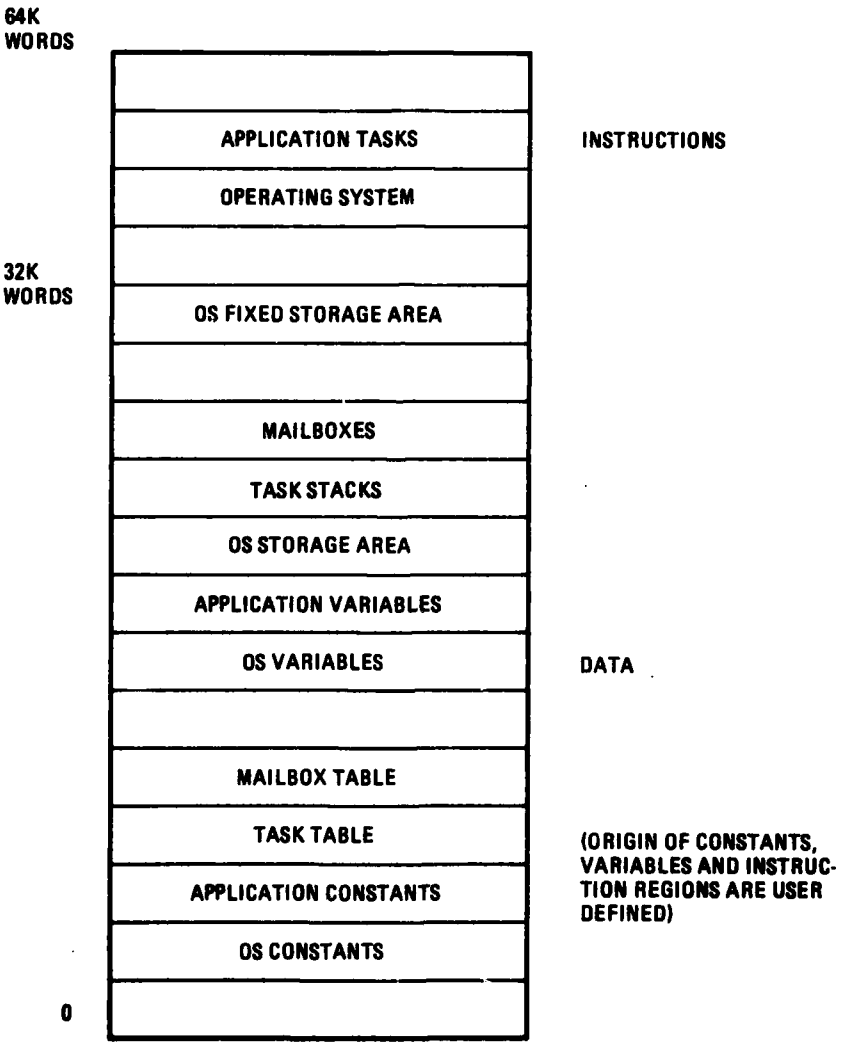


Figure 3. DIS Computer Load Pattern

If there are no defined tasks, i.e., if the program is strictly a global load, the task table, mailbox table, OS storage area, task stacks, and mailbox areas are not allocated. Under no circumstances are the four program regions allowed to overlap one another.

RESULTS

When it became apparent that the DDS JOVIAL compiler would not be available in time to meet the DLL delivery schedule, it was decided to write the DLL software in the DEC FORTRAN IV language. The lone exception is three routines within the DLLCTL CPC that enable I/O to be performed on relocatable object code modules; these are written in MACRO-11 assembly language.

As mentioned earlier in the discussion of the disk overlay structure, DLLCTL is the control CPC whose main purpose is calling the remaining five CPCs in succession. Because of this design, the CPCs were developed in the order of program execution. At each stage of testing (i.e., as coding of a new CPC was completed), all CPCs developed to that point were tested. No CPC was ever tested as an isolated unit.

The entire DLL program is a single executable task that runs within the control of the RSX-11M OS.

CONCLUSION/RECOMMENDATIONS

Some observations and recommendations concerning the linking loader program are:

a. It should be possible to specify individual modules from an object file in a LOAD directive.

b. Some directives are used so seldom or are unnecessary that it is difficult to justify the large effort it took to implement them. These are DISPLAY, OUTPUT (to a file), and REGION.

c. The execution time for DLL is often quite lengthy. A series of messages, each defining the current stage of processing, should be outputted to the operator during this "waiting period" to assure him that all is well.

d. Some error messages provide little information to the operator concerning the source of the problem. The worst offender seems to be

    ***INSUFFICIENT CORE FOR ALLOCATION - PROGRAM FILE FORMATTING.

    This particular message is output whenever one of the following is detected:

    1. Not enough core is available in the variables region for the allocation of the OS storage area, mailboxes, and task stacks.

    2. Not enough core is available in the constants region for the allocation of the task table and mailbox table.

It is evident that the wording of the diagnostics is lacking in a detailed diagnosis of the problem.

e. The list of externals output for an individual module in a Core Region Memory Allocation description should either be in alphabetical or address order.

f. Add the capability to link tasks with a previously linked operating system.

SECTION IV

DIS DIAGNOSTIC STATION (DDS) CONTROL, MONITOR, AND DIS MEMORY LOAD SOFTWARE

INTRODUCTION

Since the DIS computers do not have peripheral I/O devices or "front panels" that, in mini and large scale computers, provide user control and visibility of software operations, there was a requirement to provide these capabilities in a supporting hardware/software system. The DIS Diagnostic Station (DDS) was conceived to fulfill this requirement. A major component of the DDS software is the Control, Monitor, and DIS Memory Load (CML).

OBJECTIVE

The objective of the CML software was to provide a software development tool that would enable DIS users to load executable programs into DIS computers and to control and monitor the execution of these programs.

The CML software was to be written in the JOVIAL J73 language and operate on the DIS Diagnostic Station (DDS) computer (see Section X) within the DEC RSX-11M Operation System (OS). The user was to interface with CML via a terminal keyboard and display and CML was required to communicate with up to eight DIS computers on the DIS multiplex (DISMUX) bus.

APPROACH

The CML functional requirements were defined and are documented in specification number CS64-32111, Part I. The DISMUX bus was selected as the medium for CML communications with the DIS computers since a BIU I/O card is the only I/O card that every DIS computer is guaranteed to have. DISMUX monitor messages, commands from the DDS to DIS and responses from DIS to DDS, were designed to accomplish the basic monitor functions. These functions include start, stop, single-step, display and alter memory, display and alter registers, and transmit and receive DISMUX messages. The display and alter memory monitor functions were also used to perform the DIS memory load functions (download, save, and verify). A special option, which interfaces with a Programmable ROM (PROM) programmer instead of a DIS computer, allows a user to program Erasable Programmable ROM (EPROM) modules with data from files produced by the DIS Linking Loader (DLL).

The primary user interface with the CML software is the DDS terminal keyboard and display. Certain options send data to the Decwriter or line printer for hard copy output. The DDS cartridge disks are used for storage of DIS hardware configuration description files and DIS executable program files. A page and

menu oriented display scheme is used to provide user control and to provide for output of data. To furnish consistent user interface, the first three lines of the display screen were dedicated to specific functions: line 1 displays, prompts and accepts user inputs; line 2 is used for error message display; and line 3 is used for status reports and confidence messages. The remainder of the screen is used for data and menu display.

The CML Computer Program Configuration Item (CPCI) is segmented into seven Computer Program Components (CPCs) as shown in Figure 4.

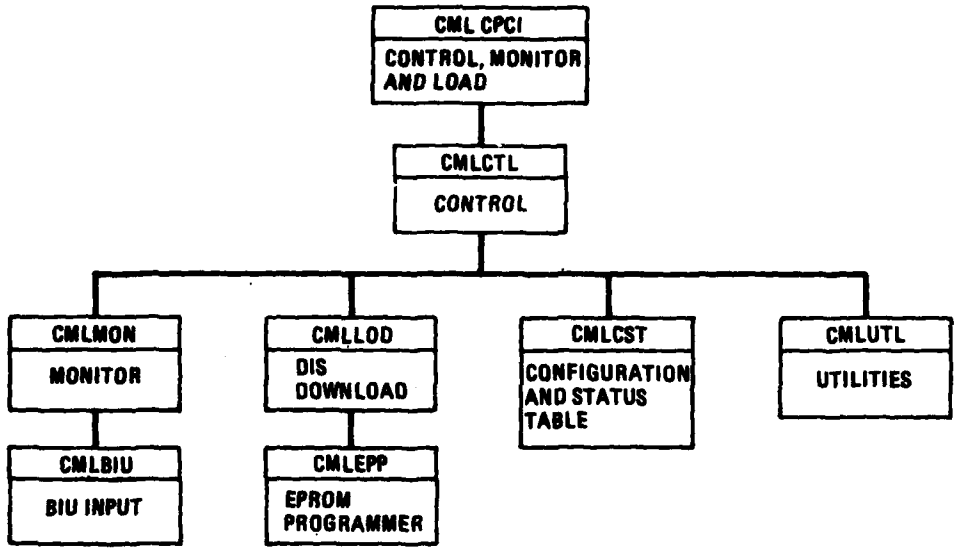


Figure 4. CML CPC Segmentation

RESULTS

When it became apparent that the DDS JOVIAL compiler would not be available in time to meet the CML delivery schedule, it was decided to write the CML software in the DEC FORTRAN IV language.

Except for CMLCTL, which is a library of utility modules used by the other CPCs, each CPC is an executable task which runs within the RSX-11M OS.

The CML software was developed in a top-down manner, with the basic menus and high level control paths being implemented first using stubs for the lower level modules. This development process eliminated the need to code "throw-away" drivers for testing lower level modules and also provided for continual testing of all module interfaces in the control path from the highest level down to the module under test.

Initial DISMUX message communication tests were monitored with a logic state analyzer. The alter and display memory monitor functions were developed first since they were required for the download process. After the basic alter and display memory functions were working, a coordinated test effort introduced the remaining monitor functional modules one-at-a-time in both CML and the DIS OS. Final integrated testing was performed with the complete CML and a multi-task DIS program.

Terminal display modules were tested first with simulated DIS data generated by stubs. When the display and operator interface function testing was complete, the stubs were replaced with the actual DISMUX I/O modules.

CONCLUSIONS/RECOMMENDATIONS

If CML is to be used in a mode where a given sequence of operations is performed many times, it might be desirable to provide a macro or indirect command file capability. Also, a mode to run CML from the Decwriter would enable the user to obtain a hard copy of the debug session. Currently CML displays BIU messages in only a hex or ASCII format. Debugging would be enhanced by allowing the user to specify the format for display of the monitored BIU messages. Also CML uses absolute memory addresses for such functions as display, memory, alter memory, and set breakpoint. The capability of symbolic references to addresses would alleviate the user from knowing the absolute memory locations.

SECTION V

DIS OPERATING SYSTEM (OS)

INTRODUCTION

An operating system (OS) is a collection of software modules that bridges the gap between a bare computer and an application programmer. Its primary purpose is to effectively and efficiently manage the hardware resources and control the software operations of a computer system. It is a logical extension of the hardware and allows the application programmer to concentrate on solving problems rather than being concerned with the hardware design, system communication I/O data formats or protocol requirements, or the internal software structure to be implemented.

Operating systems have two basic functions: they provide services for application program development and act as an environment in which application programs execute. The features of an OS are usually dependent on the requirements of the end user. For DIS applications, the primary OS features include:

a. System startup and power failure/restoration processing.

b. Program loading, executing and debugging.

c. Communication/control processing, which includes interrupt processing, task management, I/O management, and data base management.

d. Hardware fault detection and recovery processing.

These OS features control the DIS hardware and software resources by: interfacing with the DIS hardware and I/O devices, providing a software organization that standardizes the interfacing of the DIS application software modules, and supervising the execution of those modules in a multi-computer real-time environment.

OBJECTIVE

The DIS OS consists of a collection of software modules that allow the DIS programmers to concentrate on solving application problems. The OS eliminates the need for programmers to be concerned with the DIS hardware architecture (e.g., interrupts, I/O device characteristics, instruction set, memory [RAM or PROM], system startup and power failure), and system communication protocols required on the MIL-STD-1553B and DIS multiplex (DISMUX) bus data link. The OS also lessens the need for programmers to be concerned with the DIS multicomputer operational environment.

Additionally, the OS provides the capability to interface the application tasks (module is an equivalent term) into an integrated collection of software that can be configured to solve a DIS application problem. This capability is provided

by the application software executing task management statements that are processed by the OS. Those statements allow the software to detect and respond to the occurrence of events (e.g., interrupts, I/O initiated/completed, time slicing, or computation). Through the use of task management statements and the rule that tasks may not invoke other tasks directly but only indirectly through the OS, a software structure is provided that can be used to efficiently control the design and management of the DIS application software during the entire life cycle of the DIS program.

The specific OS major functions providing the previously outlined capabilities are:

a. Task Scheduler

b. I/O Management

c. Task Management

d. Interrupt Management

e. I/O Device Handlers

f. Fault Detection/Recovery

g. Data Base Management

h. Data Collection/Extraction

i. DIS Diagnostic Station Monitor Interface

A brief synopsis of each of the above functions is provided in the following paragraphs.

Task Scheduler

The DIS application tasks execute asynchronously in real time and compete, based on an "a priori" and/or dynamic, priority structure, for obtaining CPU control and the use of a computer's resources (i.e., time availability, I/O device utilization, memory, event detection, interrupt detection).

The task scheduler supervises the execution of the application tasks and controls and manages the resources. The task scheduler uses a scheduling algorithm based upon the priority and "state" (i.e. ready, suspended, dormant, or executing) of each task. The task scheduler gives CPU control to the task having the highest priority that is in the executing or ready state.

I/O Management Function

The I/O management function is the interface between the applications software and the OS for all DIS I/O operations. The I/O operations (i.e., read, write, request use of a device (open), and release a device (close)) are requested by the application tasks executing the appropriate I/O statements. The I/O

management function processes those statements and decodes and dispatches the read and write requests to the I/O device handlers function for I/O initiation. Additionally, the task scheduler is invoked for possible task rescheduling. The open and close statements are used by the OS to manage the I/O devices and buses to allow for possible multiple users.

Task Management Function

The Task Management Function processes task statements executed within the applications software. Those statements request the OS to perform the following subfunctions:

a. Initiate a task at a specified priority level.

b. Suspend a task from execution.

c. Change a task's priority.

d. Transmit an intertask message to another task.

e. Receive an intertask message.

f. Schedule a task after an elapsed time interval.

g. Schedule a task when a semaphore flag is cleared by another task.

h. Ready a task for execution.

i. Terminate a task.

j. Read real-time clock.

k. Set real-time clock.

The task management function invokes the executive function after processing most user task statements for possible task rescheduling.

Interrupt Management Function

The Interrupt Function of the OS recognizes all external and internal interrupt types. The external interrupts are a result of hardware detected events (e.g., I/O device and bus interrupts, system startup and power failure detection, occurrence of a breakpoint, single instruction completion, or interval timers time out).

I/O Device Handlers Function

This OS function contains the required I/O and device control instructions that allow data transfer to take place between a computer and an I/O device, weapons bus, or MIL-STD-1553B avionics bus. Device handlers exist for each of the DIS I/O cards (i.e., programmed I/O (serial and parallel), DMA, MIL-STD-1553B, and the DISMUX).

Each device handler accommodates any required software and hardware protocols. This allows the applications software to be transparent to the actual hardware design, I/O data formats, error detection/correcton techniques, software/hardware handshaking, etc. This transparency allows the system to accommodate any I/O changes (hardware or software) with no impact on the applications software.

Fault Detection/Recovery Function

This function detects DIS hardware malfunctions and initiates recovery action where possible.

Data Base Management Function

The DIS federated computer concept requires information to be exchanged over the DISMUX between computers. The implementation of data base management is handled by task statement calls to the OS. Those statements identify the location and size of information transmitted over the DISMUX.

Data Collection/Extraction Function

This function collects/extracts data (e.g., task timing, I/O device utilization, OS and task control tables) and transfers it for analysis/reduction.

DIS Diagnostic Station Monitor Function

This function processes the DDS monitor commands received over the DISMUX. This function and the data-collection/extraction-function are used to support the testing and validation of the DIS hardware and software.

APPROACH

The DIS computers are based on the Z8002 CPU built by Zilog, Inc. The OS design apprcach supports the Z8002 CPU features for interrupt structure, system/normal modes of operation, system trap structure, program registers, stacks (user and system), and program status. All actions of the OS are initiated by either a vectored interrupt, non-vectored interrupt, or system trap. The vectored interrupts and their priorities are shown in Table 3. The card interrupts have different functions depending upon the card plugged into the slot. The OS configures the correct interrupt service routine linkage at initialization based upon the I/O configuration table loaded as part of the executable program. The OS currently does not support any non-vectored interrupts. If a non-vectored interrupt occurs, the OS processes it as an illegal operation. System traps consist of unimplemented instruction, privileged instruction in normal mode, system call, segmentation trap, and breakpoint. The unimplemented instruction, privileged instruction, and segmentation traps are processed as illegal operations with appropriate status recorded. Breakpoint processing supports the breakpoint and single step monitor functions. System calls are

treated as software interrupts, and their initial processing to save the state of the interrupted software is the same as for vectored interrupts. The system calls supported by the OS are shown in Table 4.

TABLE 3. DIS VECTORED INTERRUPTS

Vector	Function
15 (Highest Priority)	Power Down Save Memory Command
14	Unused
13	Unused
12	Monitor
11	Card 0 Input
10	Card 0 Output
9	Card 1 Input
8	Card 1 Output
7	Card 2 Input
6	Card 2 Output
5	Card 3 Input
4	Card 3 Output
3	Type B Interval Timer
2	Type A Interval Timer
1	Unused
0	Unused

TABLE 4. SYSTEM CELL ASSIGNMENTS

SC 0	ILLEGAL INSTRUCTION
SC 1	INITIATE A TASK
SC 2	TERMINATE A TASK
SC 3	CHANGE A TASK'S PRIORITY
SC 4	DISABLE/ENABLE SCHEDULER
SC 5	DECLARE A TASK AS PERIODIC
SC 6	END BODY OF PERIODIC TASK
SC 7	DECLARE TASK AS EVENT-DRIVEN

TABLE 4. SYSTEM CELL ASSIGNMENTS (CONCLUDED)

SC 8	END BODY OF EVENT DRIVEN TASK
SC 9	SUSPEND A TASK
SC 10	RELEASE A SUSPENDED TASK
SC 11	SET REAL-TIME CLOCK
SC 12	READ REAL-TIME CLOCK
SC 13	ASSIGN ALARM TIMER 2 TO TASK
SC 14	WAIT ON SEMAPHORE FLAG
SC 15	TRANSMIT INTERTASK MESSAGE
SC 16	RECEIVE INTERTASK MESSAGE
SC 17	TRANSMIT OVER DISMUX
SC 18	RECEIVE OVER DISMUX
SC 19	ASSIGN I/O DEVICE TO TASK
SC 20	RELEASE I/O DEVICE FROM TASK
SC 21	ASSIGN 1553 OUTPUT BUFFER
SC 22	MAKE 1553 SERVICE REQUEST
SC 23	ENABLE/DISABLE 1553 MF BIT
SC 24	READ DATA FROM PIOC OR SIOC
SC 25	READ DATA FROM DMA CARD
SC 26	READ DATA FROM 1553
SC 27	WRITE DATA TO PIOC OR SIOC
SC 28	WRITE DATA TO DMA CARD
SC 29	WRITE DATA TO 1553
SC 30	WAIT FOR SPEC INTERRUPT
SC 31	REMOVE INTERRUPT WAIT
SC 32	RETURN INFO ON CPU FAILURES
SC 33	RETURN INFO ON SPEC TASK
SC 34	RETURN TIMING DATA ON TASK
SC 35	ENABLE/DISABLE DATA COLLECT

The operating system physically consists of two distinct sections – the bootstrap and non-bootstrap. The bootstrap section is absolute and physically resides in PROM in the first 2048 decimal locations of instruction memory. It contains the power up, initial trap and interrupt control, and monitor processing. Upon power up, the bootstrap initializes the computer and waits for monitor interrupts. An executable program (containing the non-bootstrap OS) can then be down-loaded (via control, monitor, and load on the DDS) and set into execution.

The non-bootstrap section is relocatable and contains the control and executive functions, the task management functions and the input/output management functions. It is linked by the DIS linking loader along with the user defined tasks. The monitor START command sets the application program (under control of the OS) into execution. The task continues to execute until it issues a system call, an interrupt, or trap. Figure 5 shows the OS processing when one of these conditions occurs. First the hardware interrupts through the program status area in the bootstrap. The bootstrap disables all but the power down interrupt and branches to the non-bootstrap (in RAM) to save the state of the interrupted software. The OS enables higher level interrupts to support interrupt nesting. Depending on whether a system call or interrupt occurred, the appropriate handling routine is invoked. It may alter the task priority table. After the service processing is completed, the task scheduler (if required) is executed to determine the task eligible for execution. The previous level of interrupts is restored and the eligible task is set into execution until another system call or interrupt.

RESULTS

The DIS Operating System Computer Program Configuration Item (CPCI) is allocated to four Computer Program Components (CPCs) – the Executive and Control CPC, the Task Management CPC, the Input/Output Management CPC and the Monitor CPC. These CPCs are each a collection of separately assembled modules that reside in a library file (except for the bootstrap modules). The library file is loaded along with applications tasks by the DIS Linking Loader to produce an executable program for a DIS computer. The operating system modules are listed in Table 5. They support the interrupt handling and system call functions (Table 4).

The memory storage requirements (not including the PROM bootstrap) for the entire operating system are:

Instructions:	5143 locations
Constants:	OS data base = 176 Task table = 5 per task + 1 Mailbox identification = 96

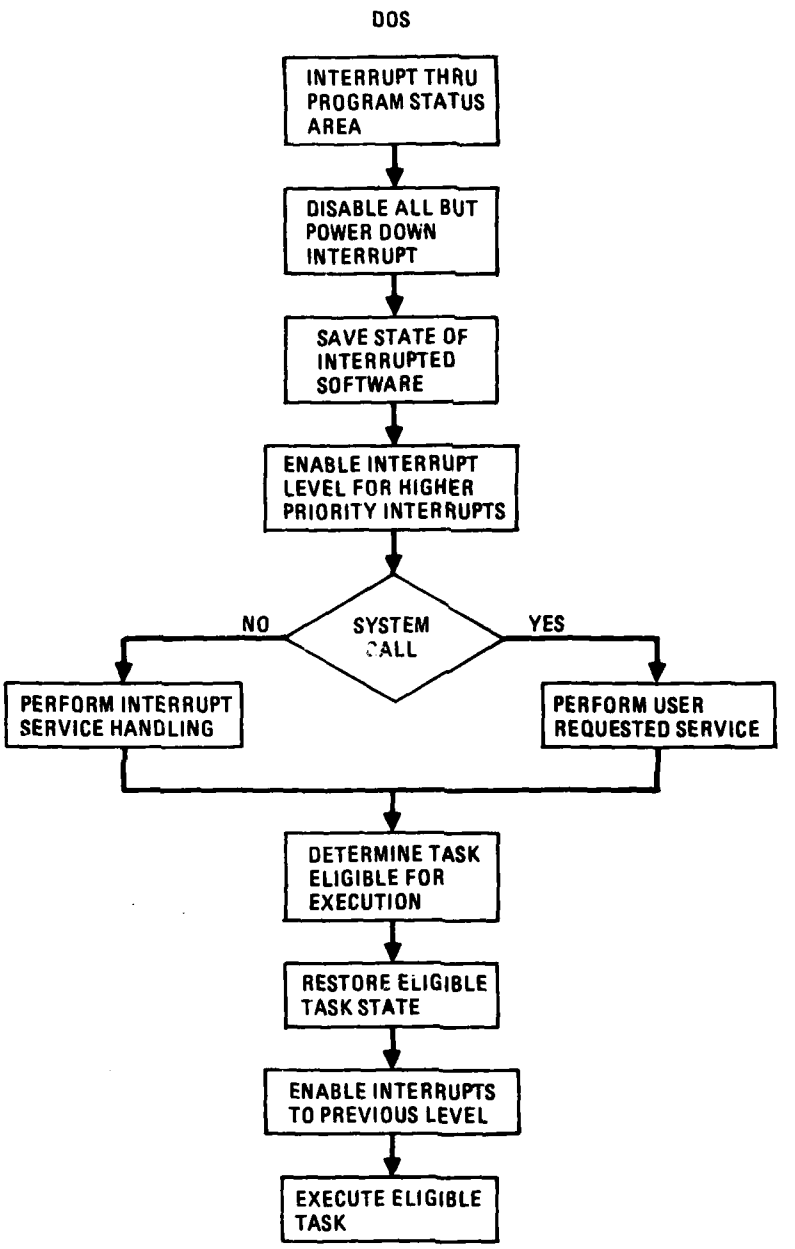


Figure 5. DOS Functional Flow

TABLE 5. DIS OPERATING SYSTEM MODULES

Executive and Control CPC

- DBVAR Data Base Variables
- DBCON Data Base Constants
- INTCTL Bootstrap PROM Interrupt Control
- CPUTST DIS CPU Checksum and Instruction Test
- OSCNTL DIS Operating System Interrupt Control
- OSINIT DIS Operating System Initialization
- CPUST CPU System Status Service
- TASKST Task Status System Service
- TASKTM Task Timing System Service
- DATCOL Data Collection System Service
- EVTREC Data Collection Event Recording
- CLKCTL Real-Time Clock Control
- DDCNTL Data Collection Utility Routines
- TIMING Task Timing Maintenance

Task Management CPU

- PERIOD Periodic and End-Periodic System Service
- CLOCK Set Clock and Read Clock System Service
- ALARM Alarm System Service
- INTCON Interrupt and Disconnect System Service
- TASK Initiate Task
- TKMUTL Task Management System Call Utilities
- XMTREC Transmit and Receive Intertask Message
- SUSREL Suspend and Release Task
- EDSCHD Enable/Disable Scheduler
- SEMAPH Wait on Semaphore
- PRIRTY Change Task Priority
- EVSTRT Signify Start and Termination of Event Driven Task
- ABORT Abort Task

TABLE 5. DIS OPERATING SYSTEM MODULES (CONCLUDED)

Input/Output Management CPC

• RBIU	Receive Over DISMUX
• XBIU	Transmit Over DISMUX
• RDMA	Read From DMA Card
• WDMA	Write to DMA Card
• A1553	Assign 1553 Output Buffer
• S1553	Set 1553 Service Request Bit
• MSLFLT	Enable/Disable 1553 Missile Fault Bit
• R1553	Read From 1553
• W1553	Write to 1553
• READPS	Read Parallel/Serial Device Input Data
• PSUTL	Parallel/Serial I/O Utilities
• WRITPS	Write Parallel/Serial Output Data
• OPEN	Open I/O Device
• CLOSE	Close I/O Device
• DINUTL	DMA Input Utilities
• INTUTL	Interrupt Notification Utilities
• DOUTUT	DMA Output Utilities
• XOVRTM	DMA Transmit Overtime Check
• DMAUTL	DMA Management Utilities
• 01553U	1553 Output Handler Utilities

Monitor

• MONITOR	Bootstrap PROM DDS Monitor Interface
• DOSBSC	Bootstrap Configuration Identification

Variables:
OS data base = 797
OS task storage = 54 per task
Task stacks = user specified
OS fixed storage = 460

Mailboxes: NULL = 257
DMA = 257 per mailbox
BIU = 35 per mailbox
1553 = 35 per mailbox

The bootstrap portion of the OS resides in PROM in the first 2048 memory locations of instruction space. Approximately 1550 locations in the PROM are used by the OS.

One of the design goals for the OS was to keep the functions modular to enable the OS to be configured depending upon the application and input/output mix. The memory requirements to support the minimum OS functions of processing interrupts and system calls, providing input and output via the DISMUX, executing event-driven and periodic tasks, and servicing the real-time clock are:

Instructions: 2234 locations

Constants: OS data base = 176
Task table = 5 per task + 1
Mailbox identification = 96

Variables: OS data base = 283
OS task storage = 54 per task
Task stacks = user specified
OS fixed storage = 460
Mailboxes - NULL = 257
BIU = 35 per mail box

CONCLUSION/RECOMMENDATIONS

The DIS OS met its objective of supporting real-time multi-tasking software applications in a multi-computer environment. Its design and implementation for the federated computer concept are adequate for a research and development program. In its current configuration, it provides a good baseline for the development of a more optimal operating system for full scale engineering development (FSED). Its modular design and implementation facilitate the addition or deletion of functions. Interrupt processing and task scheduling should be studied to reduce system timing overhead. Most of the system table initialization and input/output configuration should be performed by the linker or loader programs to reduce the OS memory requirements. Also conditional coding of parameter checking, data collection, and status return processing would allow these features to be utilized during initial program debug, but removed from final program production for flight mode.

SECTION VI

DIS/DDS DIAGNOSTICS SOFTWARE

INTRODUCTION

The degree of sophistication inherent in the Digital Integrating Subsystem (DIS) and DIS Diagnostic Station (DDS) computers requires a software method by which the individual hardware components can be tested should problems occur. To a certain extent the DIS/DDS Diagnostics Software (DDP) is the means by which this requirement is met.

This section presents various aspects of the DDP design philosophy, concluding with suggestions and recommendations for program enhancement.

OBJECTIVE

The purpose of the DIS/DDS Diagnostics Software program is to provide the DDS operator with tests that allow him to detect and isolate problems associated with a single DIS computer (DIS DIAGNOSTICS) and problems associated with the DDS I/O cards that interface the DDS to the DIS (DDS DIAGNOSTICS).

APPROACH

The DIS/DDS Diagnostics Software consists of three major functions: DDS-DIS DIAGNOSTICS, DIS-DIS DIAGNOSTICS, and DDS DIAGNOSTICS.

DDS-DIS DIAGNOSTICS include two distinct tasks of DIS DIAGNOSTICS software resident (not simultaneously) in the DDS computer (PDP 11-34) that operate with the DIS-DIS DIAGNOSTICS.

DIS-DIS DIAGNOSTICS include two distinct groupings of DIS DIAGNOSTICS software resident (not simultaneously) in the DIS computer. This software, together with the DIS Operating System, is downloaded from the DDS and operates with the DDS-DIS DIAGNOSTICS.

DDS DIAGNOSTICS are resident in the DDS computer (not simultaneously with DDS-DIS DIAGNOSTICS).

Both the DDS DIAGNOSTICS and DIS DIAGNOSTICS programs are initiated from the DDS Control, Monitor, and Load function, the latter after the corresponding DIS-DIS DIAGNOSTICS tasks have been downloaded into the DIS computer.

A list of the processing capabilities available within each of the three functions described above follows; a more complete description may be found in the Computer Program Development Specification for DIS Diagnostic Station Diagnostic Software, Part 1 (CS64-32112).

The following is a summary of the processing capabilities of the DDS DIAGNOSTICS program:

a. Provide for a Decwriter hard copy option.

b. Provide for the selection of any combination of up to 18 DDS I/O cards on which self-tests are to be performed. These tests consist of:

    1. BIU I/O card self-test
    2. DMA I/O card self-tests
    3. Parallel I/O card self-tests
    4. Serial I/O card self-tests
    5. 1553 I/O card self-test

c. Execute a test for each DDS card selected.

d. Output the result of each test.

The following is a summary of the processing capabilities of the DIS-DIS DIAGNOSTICS non-I/O test program:

a. Provide for a Decwriter hard copy option.

b. Accept user input of the BIU ID of the unit under test.

c. Accept user input specifying the test mode.

d. Accept user input(s) defining the test and/or required parameters. The following tests are available:

    1. BIU BOM accept/reject test
    2. Memory check test
    3. BIU supervisor EOT search test
    4. Clock/delay test
    5. CPU test

e. Execute the selected test in the specified mode (all tests executed if automatic mode specified).

f. Output the test results.

g. Detect abnormal conditions and output corresponding error messages.

The following is a summary of the processing capabilities of the DDS-DIS DIAGNOSTICS I/O test program:

a. Provide for a Decwriter hard copy option.

b. Accept user inputs defining various BIU parameters.

c. Output the I/O card configuration for the unit under test.

d. Accept user input specifying the test mode.

e. Accept user input specifying the DIS slot number of the card to be tested (not applicable in automatic mode).

f. Accept user input(s) specifying the DDS card slots corresponding to any DIS DMA, parallel, or serial cards being tested.

g. Execute the test for the selected DIS I/O card in the specified mode (all DIS cards tested if automatic mode specified). Subject to the I/O card configuration, the following tests are available:

   1. BIU I/O card loopback test
   2. DMA I/O card (peripheral-initiated input) loopback test
   3. Parallel I/O card loopback test
   4. Serial I/O card loopback test
   5. 1553 I/O card loopback test

h. Output the test results.

i. Detect abnormal conditions and output corresponding error messages.

RESULTS

When it became apparent that the JOVIAL compilers would not be available in time to meet the DDP delivery schedule, the project was temporarily shelved. When the DIS JOVIAL compiler became functional, the DDP software effort was revived, with the DDS and DDS-DIS DIAGNOSTICS to be written in the Digital Equipment Corporation (DEC) FORTRAN IV language and the DIS-DIS DIAGNOSTICS in the JOVIAL (J73) language.

The DDS DIAGNOSTICS program and both modules of DDS-DIS DIAGNOSTICS were tested by initially using "stubs" in place of the routines that actually perform the I/O. In this manner problems involving the user-program interface and test result messages were resolved before the more elusive bugs involving I/O were tackled.

CONCLUSION/RECOMMENDATIONS

Diagnostic software can be produced with a virtually limitless degree of sophistication. Only major tests were implemented in the DIS contract, demonstrating the feasibility of such a set of diagnostics.

More work needs to be performed in this area. Consideration should be given to developing automated diagnostic routines that could be adaptable for use in a depot by relatively limited-skill personnel. This type of diagnostic would be extremely useful in reducing the life cycle maintenance costs associated with the computer system in a weapon.

SECTION VII

CLASS I/II COMPUTER

INTRODUCTION

Today's airborne computers must be compact, lightweight, low in power consumption, reliable under severe environmental conditions, and capable of doing moderate to high speed computations. In the past, commercially available computers failed to meet these requirements by wide margins in all aspects except for processing throughput. The proliferation of many computers in the mid 1960s and early 1970s came close to meeting aircraft and missile requirements. Because of the unavailability of off-the-shelf computers, aircraft and missile manufacturers designed, developed, and built special purpose devices to meet their immediate needs. By building these special purpose computers, the manufacturers fell into the trap of having new software developed. Software maintenance costs have proven over the life cycle of the equipment to be greater than those of the hardware.

In recent years, major advances in large scale integration (LSI) technology have created faster, smaller, and lower priced computers. One of these advances was in the development of metal oxide silicon/large scale integration (MOS/LSI) microprocessors, such as the Zilog Z8002. This 16-bit microprocessor is 5 to 10 times faster than any 8-bit microprocessor and 2 to 5 times faster than most other 16-bit devices including popular minicomputers such as the DEC PDP 11/34.

The Z8002 processor, although primarily supplied by Zilog, is second-sourced by AMD. The second-sourcing agreement is significant for military applications for present and future needs since it will provide another low cost, reliable source. This section incorporates the results of this contract in regards to the DIS airborne computer.

OBJECTIVE

The overall objective of this contract is to develop a federated computer hardware system for use in both advanced missile applications and other airborne military applications. This system incorporates up to 32 small, low-cost computers (with their respective operating systems), interconnected by a pair of twisted shielded wires, into a distributed processing system capable of handling all data processing requirements of the airborne application.

Various operating systems and application programs in development by various sources are necessary to accomplish functions such as digital autopilot, navigation, seeker data processing, and terrain contour matching (TERCOM). Coupled with the hardware of this effort, the total system provides to the military user a small, low-cost, and efficient computer system.

The final objective of this computer is that it be packaged in less than 150 cubic inches, weigh less than 6 pounds, and consume less than 50 watts of standard vehicle +28 volt power. It is to be programmed initially in the Air Force recognized high order language, JOVIAL J73. This does not preclude use of programming in assembly language or future high order languages such as ADA. This federated computer system concept differs from other computer architecture such as the centralized system or a distributed system. The centralized system uses a single processor where all inputs and outputs, sensors, and checkout points are tied together. The centralized system has characteristics that are not desirable for most missile applications. Some of these characteristics include high modification costs, generally sole-source procurement, synchronous operation, point-to-point wiring, significant hardware and software redesign for added or changed tasks, and complex checkout procedures for the single CPU.

The distributive system is generally affiliated with a master/slave type of architecture. A familiar military architecture of the distributed system is MIL-STD-1553B. In this type of system, a master computer is tied to slave computers or terminals via a common bus. Distributive does offer significant advantages over the centralized system. The characteristics of a distributed system include reduced wiring, moderate modification or recurring costs, and potentially a more complex software package requirement for execution of dynamic task allocation, dynamic control, and bit check.

The federated computer system as used here differs from a distributive system in that there is no master/slave command response protocol among the processors. Each processor is capable of operating on independent asynchronous tasks with a potential of lowest modification costs. It is relatively easy to add or delete tasks because of the standard modular interfaces in and out of individual CPUs. Checkout of the system can be provided by directly monitoring the federated bus.

In addition to completing the development of the federated computer system hardware, this contract attempted to make further advances in establishing standards and specifications for cost effective computer design. This included the testing of the actual hardware as well as packaging of the hardware to meet identified military needs. Downstream, this hardware is to be demonstrated and verified to the established specifications and integrated into various systems such as Lear-Siegler's low-cost inertial guidance system (LCIGS) and McDonnell Douglas' unaided tactical guidance (UTG) software. The federated concept hopefully, after demonstration, will show that at a production rate, the concept is cost effective.

The primary hardware drivers for the federated computer system included no new chip development to ensure multiple sourcing and low cost. The architecture is such that modular memories and modular input/output (I/O) are interchangeable and tailorable to the application. Potential low-cost production is a primary concern since a low-cost system is highly desirable in military

applications. Also the power and weight requirements of the federated computer must be met without using hybrid circuits to further ensure multiple sourcing and low cost.

The federated computer system software drivers include eventual programming in JOVIAL J73 written in relocatable software modules for ease of mobility. Eventually all application programs and diagnostic software would also be written in JOVIAL J73, although their earlier stages may be assembly languages of either the Zilog machine or the diagnostic station (DEC 11/34).

The federated computer hardware is to be integrated into a system that allows distributed asynchronous processing such as that shown in Figure 6. This figure shows three federated computers tied together via a single bus that provides the computer-to-computer link. Each federated computer communicates with its respective I/O, operates its task software, with its operating system controlling tasking and scheduling. Information required from one computer to another is transferred via the multiplex bus at a one megabit per second rate. This system potentially offers the lowest life cycle cost and the lowest modification costs to be tailored for mission requirements. Seekers, data links, warheads, etc., can be linked together via the single twisted shielded pair of the multiplex bus cable. As vehicle complexity continues to grow to meet various mission requirements, the number of federated computers can also grow. Individual I/O changes affect only that particular computer and not the entire system as in a centralized computer architecture.

The DIS computers include a Class I 350 thousand operations per second (350 KOPS) computer and a Class II (525 KOPS) computer. Breadboarded computers were designed, fabricated, debugged and then packaged using multilayered circuit card assemblies and other complex high density avionics, to provide small lightweight airborne brassboard computers. After the manufacturing and debugging of the brassboard computers have been completed, acceptance testing of all units and qualification testing of a particular unit were also completed. The breadboard computers consist of a wirewrapped backplane and chassis, a power supply, CPU card, memory card, I/O controller card, and four card slots capable of accepting any mix and match combination of the developed I/O cards. These I/O cards include the bus interface unit (BIU), MIL-STD-1553B, parallel program handshake, serial program handshake, and direct memory access I/O cards. The addition of other I/Os to meet external computer interface requirements is not precluded assuming the internal architecture is maintained. Additional I/Os already developed on other programs include an analog-to-digital card, a digital-to-analog card, a discrete input/output card, a MK 117 card (serial link), a radar altimeter card, a tape recorder driver card (used for test flights), and an RS 232 card (used for testing and developing on this contract).

Software designers have written I/O card drivers to allow a functional checkout of each card using the diagnostic station and supporting test hardware and software. Additional test facilities were also developed to aid in this checkout. A

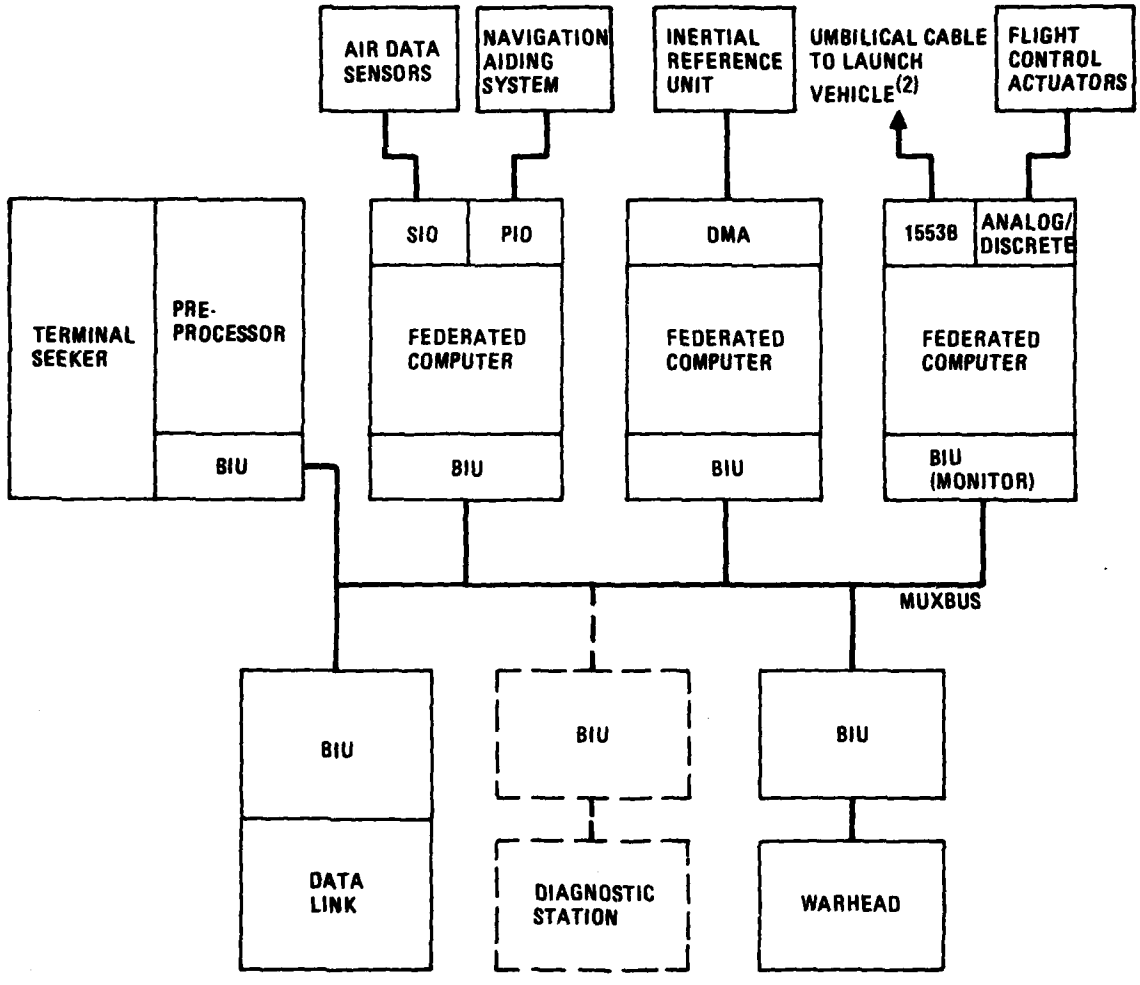


Figure 6. Typical Federated Concept(1)

task also identified for this contract was to functionally integrate and check out the federated computer hardware with the DIS Diagnostic Station (DDS) at all I/O interchange levels. This would then allow the verification of all diagnostics with the federated computer.

APPROACH

There are two versions of the airborne computer, a Class I and a Class II. The only difference between the two versions is that the Class II computer has a

faster CPU and memory card than the Class I version. All other cards are identical in both computers.

Prior to the brassboard federated computer for either Class I or Class II, a wirewrapped breadboard version was first developed and verified. Upon verification of circuit designs and software modules, the breadboard computer of both the Class I and Class II were then transformed into the multilayer printed circuit card assemblies of the brassboard.

Class I and Class II Breadboard Computer

The breadboard versions of the Class I or Class II federated computers were developed at both the General Dynamics Pomona and Convair facilities. These breadboards were built on standard vendor-supplied wirewrapped cards and card cages supplied by Garry Corporation. These physically large wirewrap versions of the federated computer allow verification of hardware and software prior to a printed circuit card commitment. Figure 7 shows the internal electrical card architecture for both the breadboard and the brassboard. Figure 8 shows the Class I CPU breadboard.

The breadboard memory card can take two forms: either a small memory or a large memory. The small memory is capable of operating in either a Class I or Class II computer, while the larger memory can operate only on a Class I computer due to its slower access time. The design goals for the memory in the federated computer were to have enough memory for tactical missile applications, very low power and to support DMA capability. Both memories have to be low enough in power so that they are capable of accepting battery backup in case of loss of primary power. The small memory is based around the Hitachi 6147 high speed CMOS RAM. This RAM is a 4K by 1 bit memory chip that can maintain its data even if the supply voltage is reduced to 2 volts. The large memory is based on the Harris 6504 CMOS RAM and their 6564 module package. This package contains 16 individual 6504 CMOS RAM chips to provide a total of 58K words of memory. The small memory using the less dense Hitachi chip provides 16K words of RAM memory. The memory card can be accessed in either words or bytes as programmed by the Z8002 processor.

The I/O controller card in breadboard form is implemented in a single wirewrap card populated with approximately 90 ICs. The I/O controller card is given the task of interfacing up to four I/O cards in any mix or match combination to the CPU/memory bus. The I/O controller is common to either Class I or Class II. Each breadboard version of an I/O card was implemented on a single Garry wirewrap card. I/O card slots within the card cage were provided for mix and match combinations of breadboard I/Os to validate hardware and software.

Checkout and debug of the breadboard computers were accomplished by using the diagnostic station and/or an RS232 monitor card to provide input. The diagnostic station allows the user to input and monitor the breadboard computer by stimulating any of the I/Os and providing a link to download and program

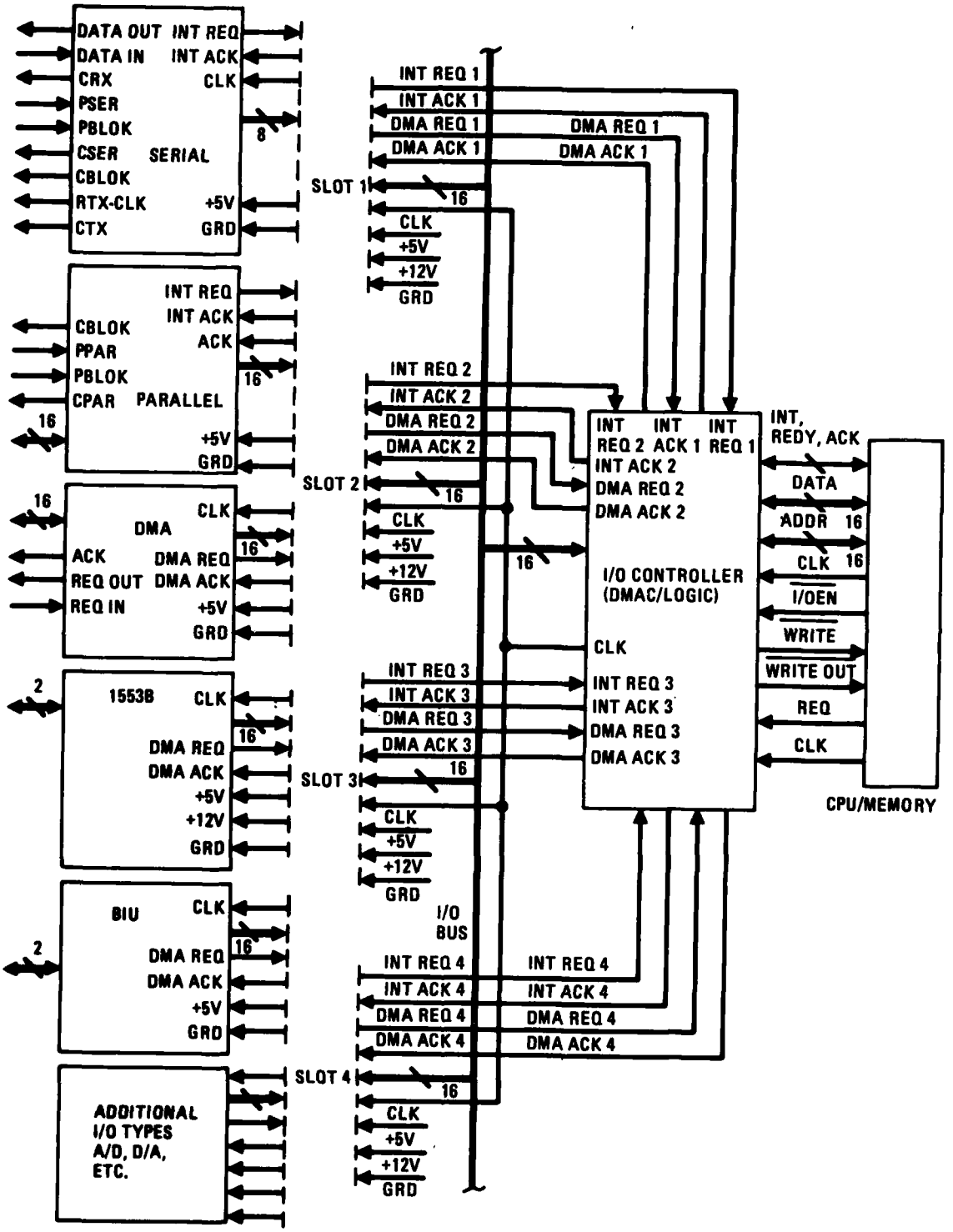


Figure 7. Federated Computer Architecture

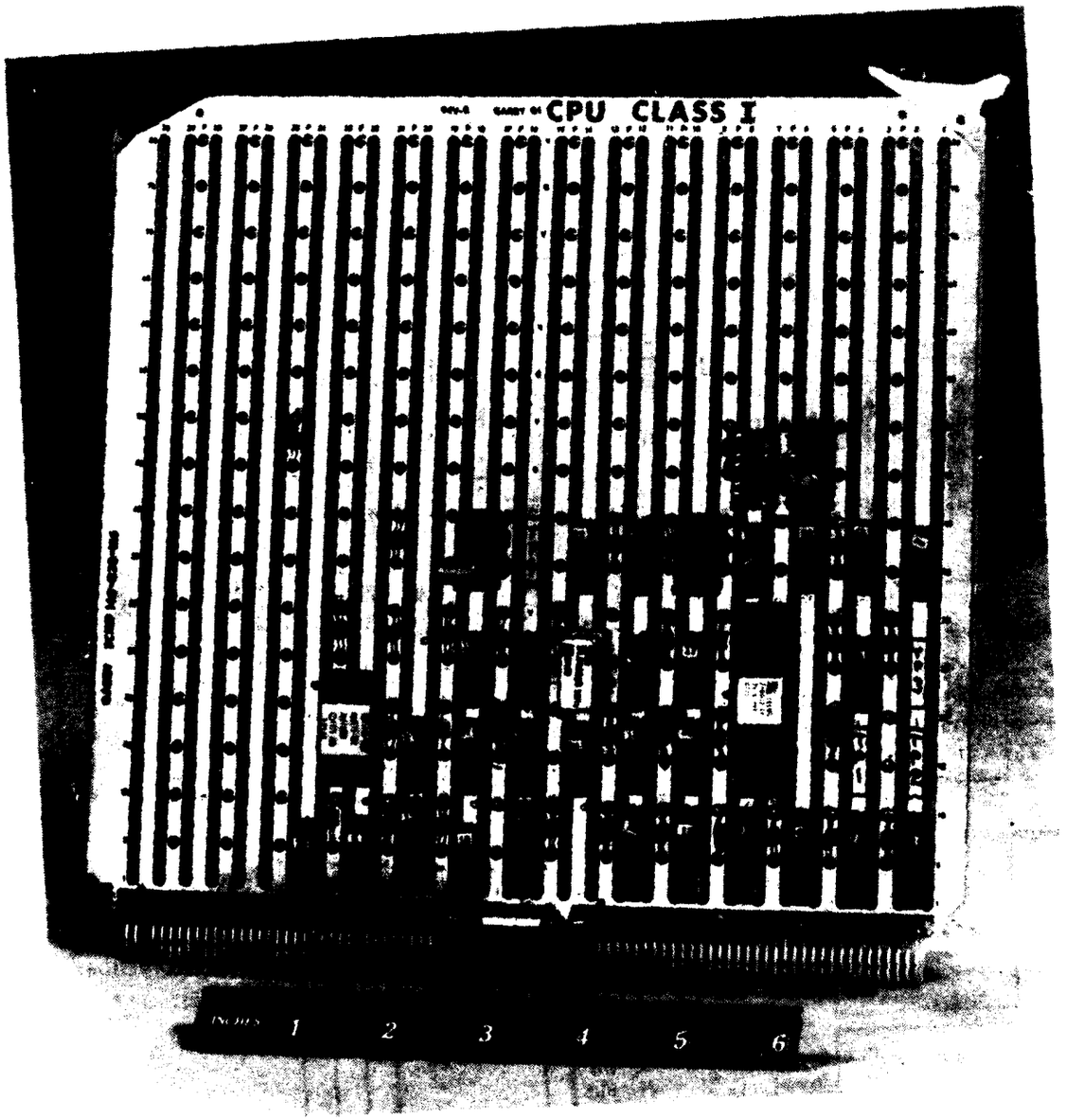


Figure 8. Class I CPU Breadboard

the computer. The RS 232 monitor card (developed on this contract) provides a simple means of executing small programs to validate proper operation of the breadboard computer under test.

The breadboard computer offers a means of validating designs and software of future I/O cards at low risk and low cost prior to a printed wiring card development.

RESULTS

Class I and Class II Brassborads

As in the Class I and Class II breadboards, the brassboard federated computer hardware has the same electrical architecture as Figure 7. The logical designs of the brassboard cards are identical to the breadboard cards; however, the physical size of the cards is extremely different.

Figure 9 shows the brassboard version of the CPU card. It is packaged on a single 24 square inch card size that is common to all other cards except for the double cards. The CPU card is configured to a Class I or Class II card by changing the clock speed with jumper wires. The backplane connector is provided by a NAFI connector at the top of the card.

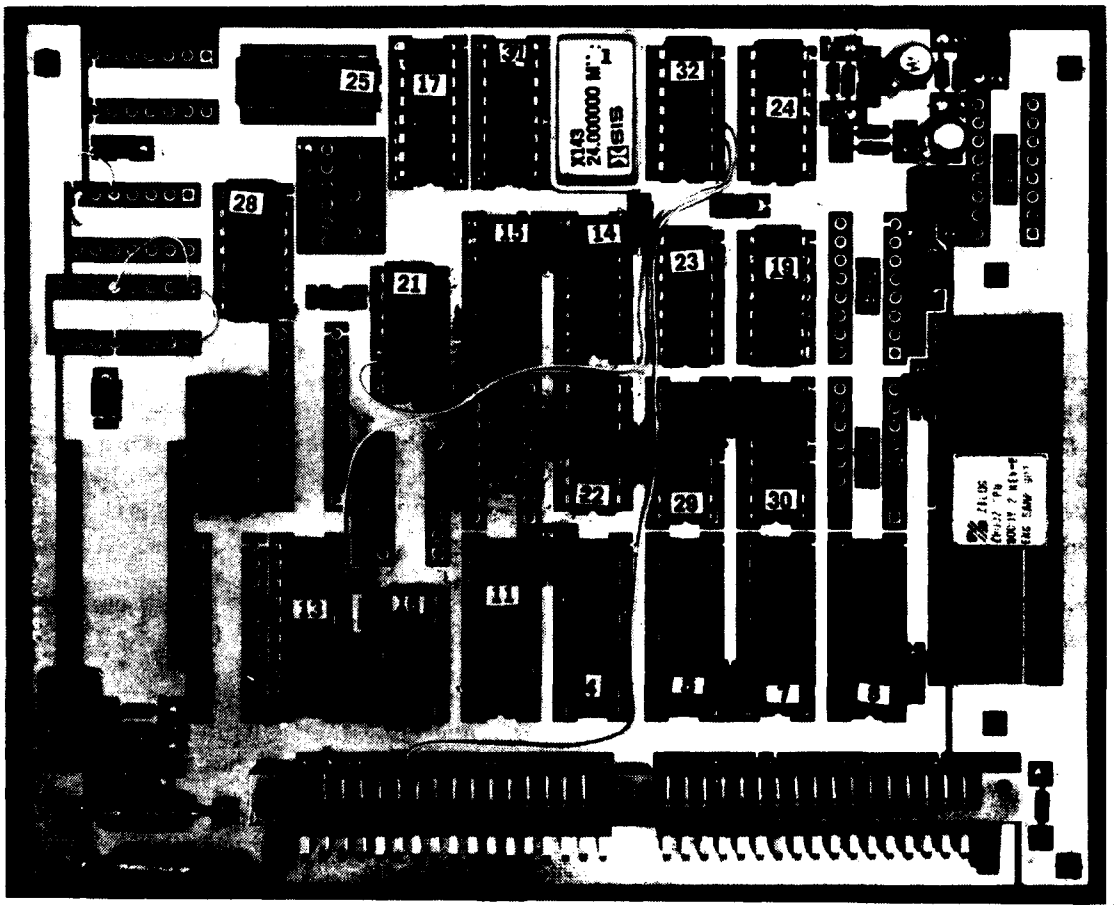


Figure 9. CPU Brassboard

The brassboard memory card is a double card interconnected with a piece of integral flex. Figure 10 shows the small brassboard memory card populated with the Hitachi memory chips. Figure 11 shows the large memory that contains the Harris memory modules. The backplane connector can be seen at one edge of either brassboard memory. Both memories fold on themselves to provide effectively a double thickness single card that has physically the identical plan view outer dimensions as a single card.

The brassboard version of the I/O controller is also a double card interconnected with a similar type of integral flex and is shown in Figure 12. This card is by far the most complicated card in terms of package density and chip count.

Backplane connections are provided via two NAFI connectors on either side of the I/O controller card and can be seen on the left and right edges of Figure 12. The I/O controller is populated with mostly flat pack IC packages, which add to the density of the card packages but add not only to assembly cost but to the expense of card checkout because of the difficulties related to connecting test probes to flat pack packages.

The I/O cards have all been packaged in the same form factor, backplane connector located at the top and its own internal unique external connector located at the bottom. Figure 13 shows the DMA I/O card, which is by far the simplest of all brassboard cards. Figure 14 shows a serial I/O card, again with backplane connector located at the top and its own unique external connector located at the bottom. The 1553B card and the BIU card are the only I/O cards containing flat packs because of their increased package density.

Figure 15 shows the brassboard backplane or flex harness. It provides the mother board or backplane connections where all circuit cards are connected. At the bottom of Figure 15, the power supply connector can be seen along with two connectors, one for the CPU and one for the memory. The long double connector is provided for the I/O controller and the four identical connectors at the top are the four slots for the I/O.

Figure 16 shows the external housing of the brassboard. The external connectors are the I/O connections and the power supply connector.

Figure 17 shows how each individual circuit card assembly is packaged into the housing. All I/O cards and the power supply are contained within the 150 cubic inch requirement. Figure 18 shows brassboard assemblies and housing. Figure 19 shows the brassboard computer filled with cards with the front cover removed to show installation.

CONCLUSIONS/RECOMMENDATIONS

This contract activity completed the objectives of verifying the breadboard and brassboard versions of the Class I and Class II federated computer. Testing and validation have shown that by taking a Class I design and increasing the

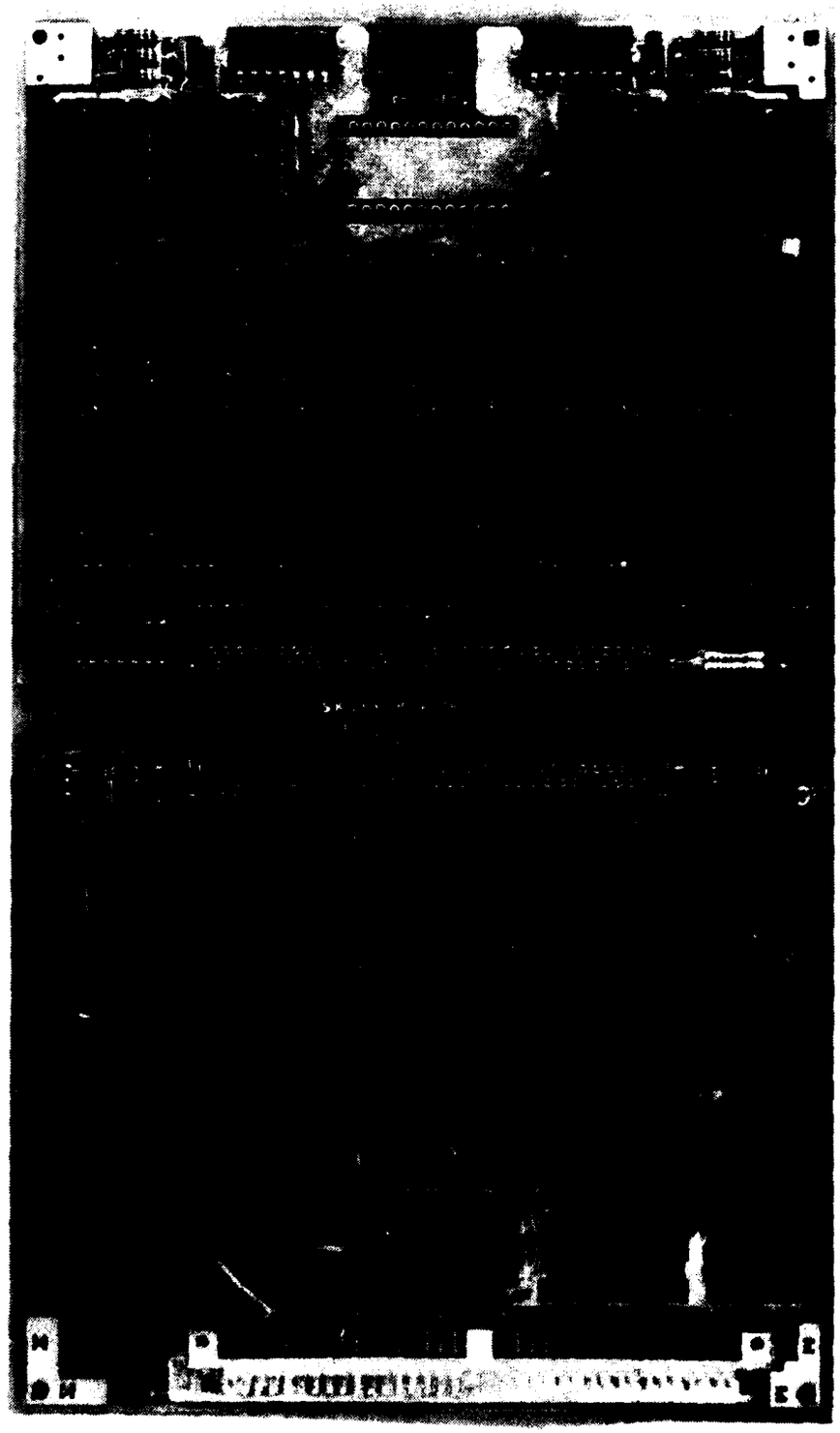


Figure 10. Small Brassboard Memory

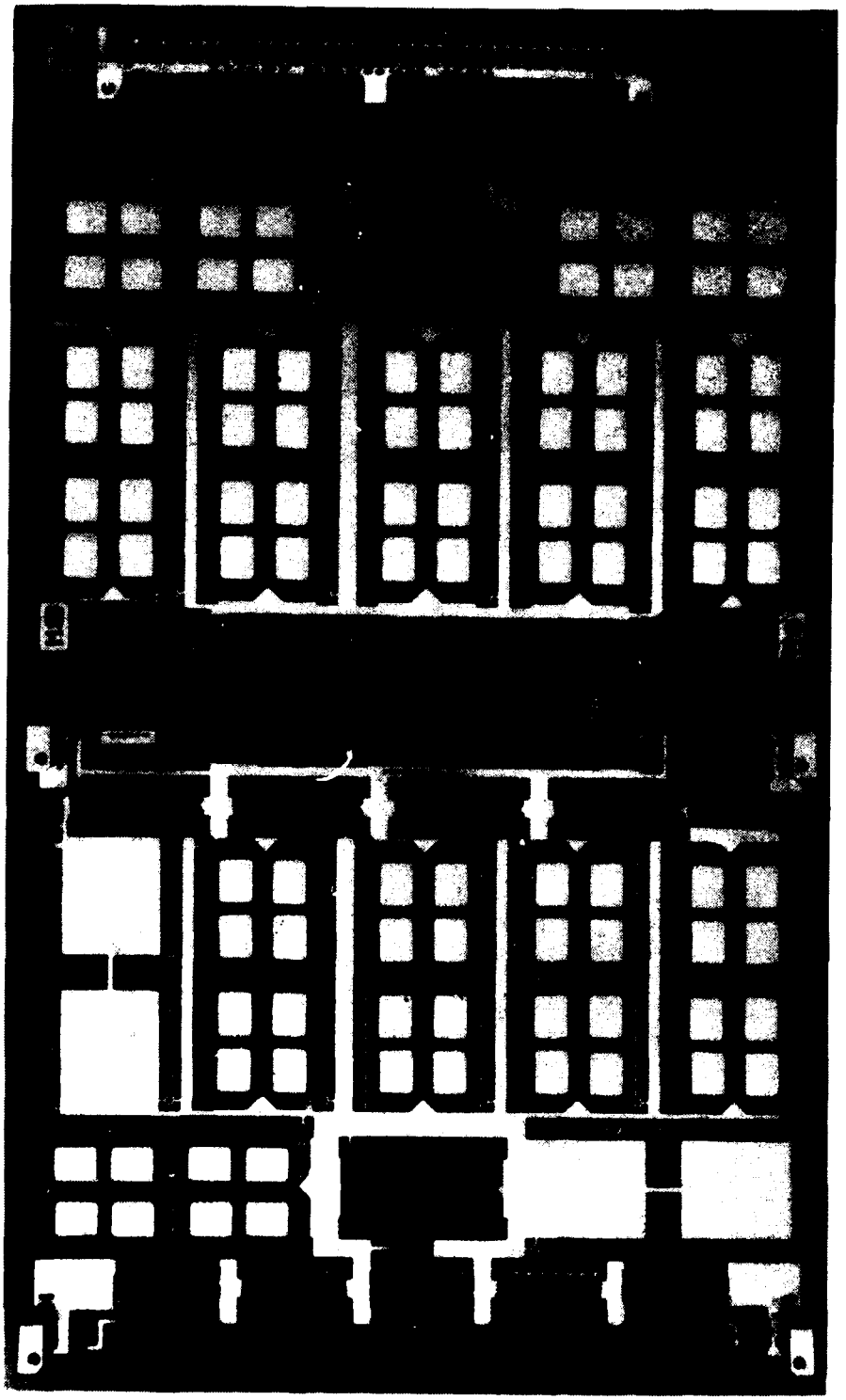


Figure 11. Large Brassboard Memory

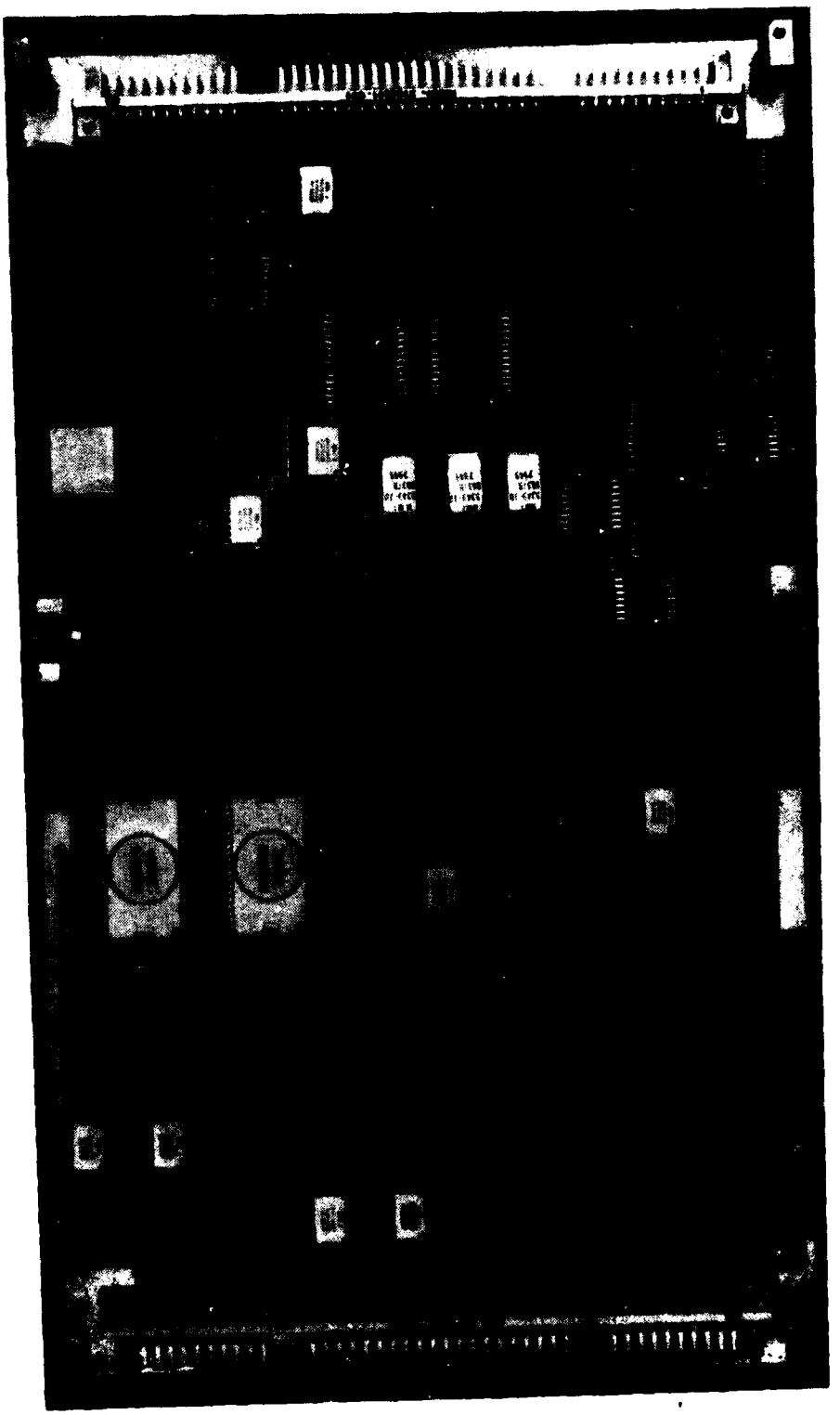


Figure 12. I/O Controller Brassboard

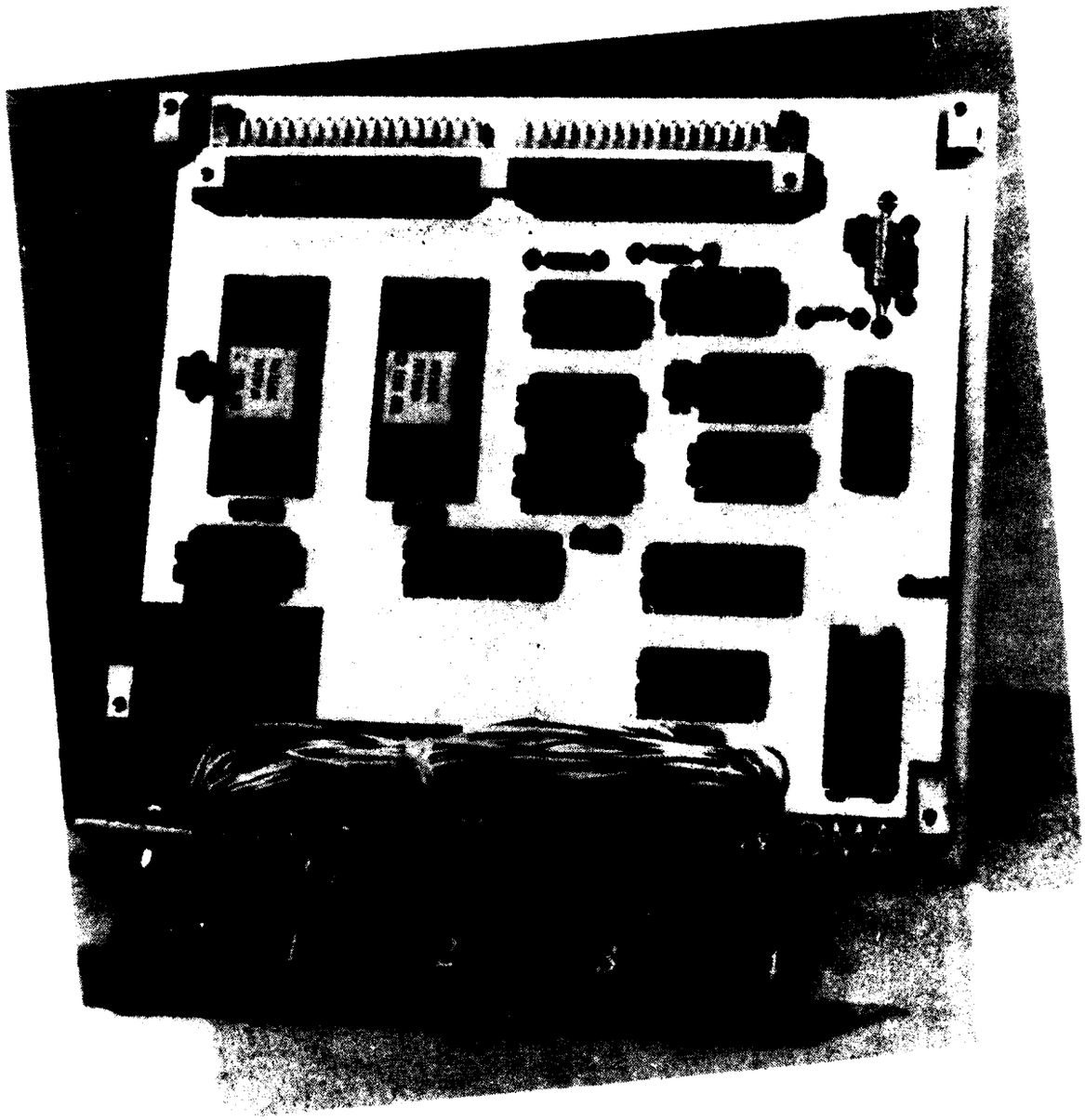


Figure 13. DMA I/O Brassboard

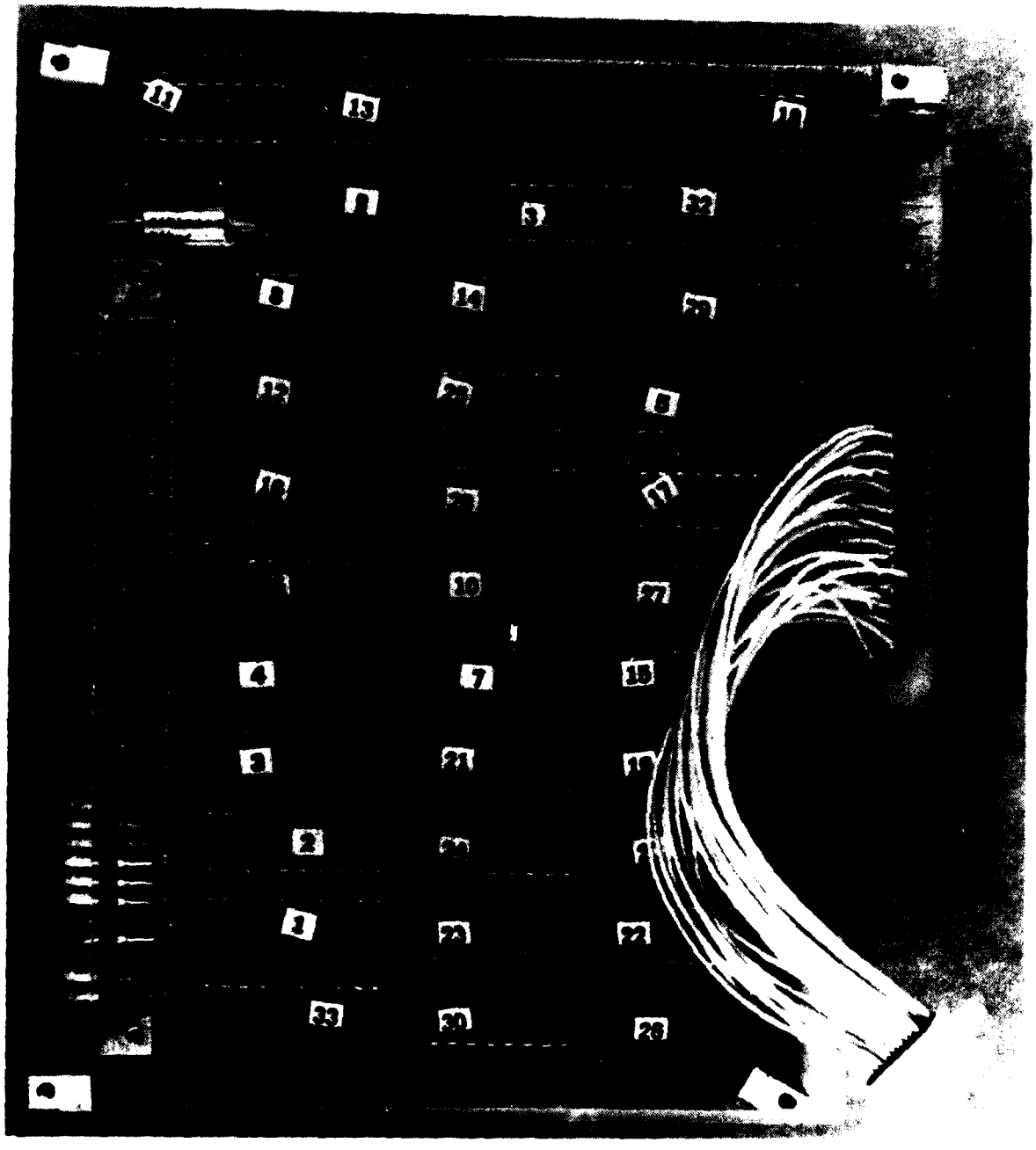


Figure 14. Serial I/O Brassboard

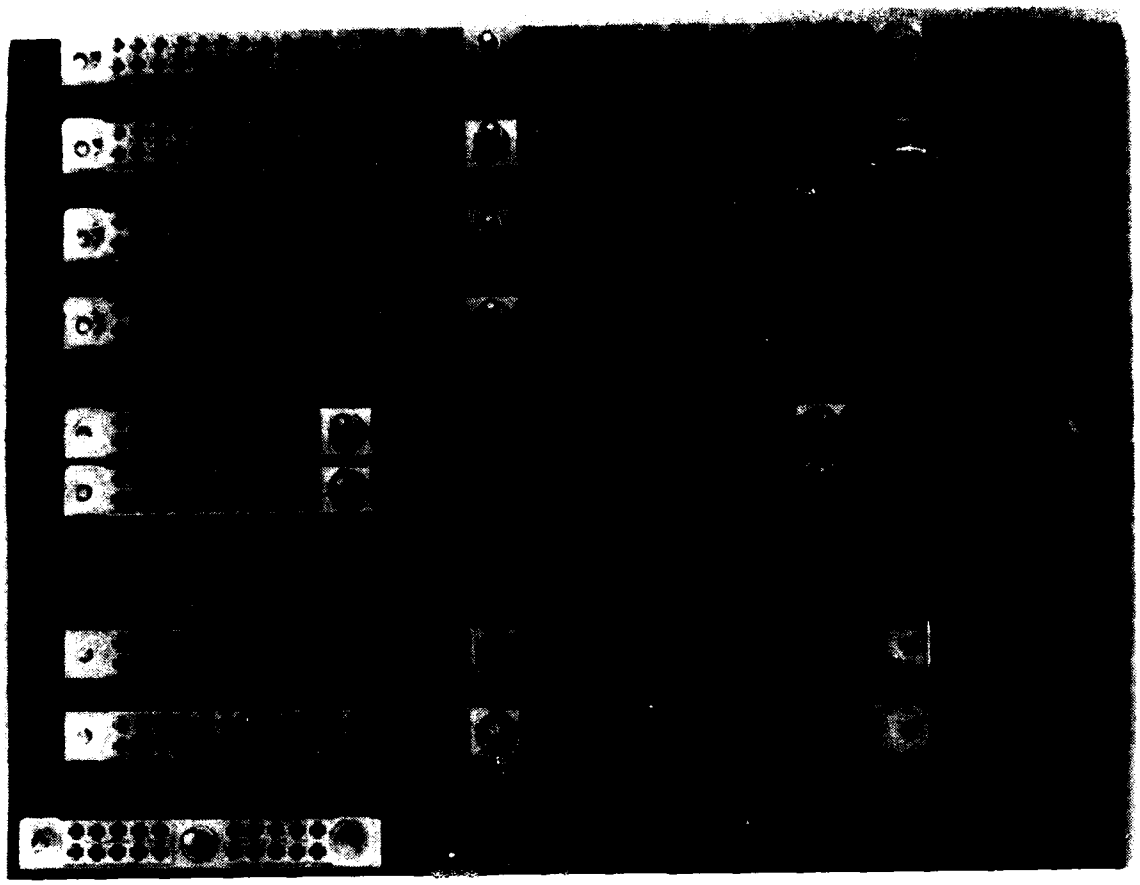


Figure 15. Brassboard Backplane

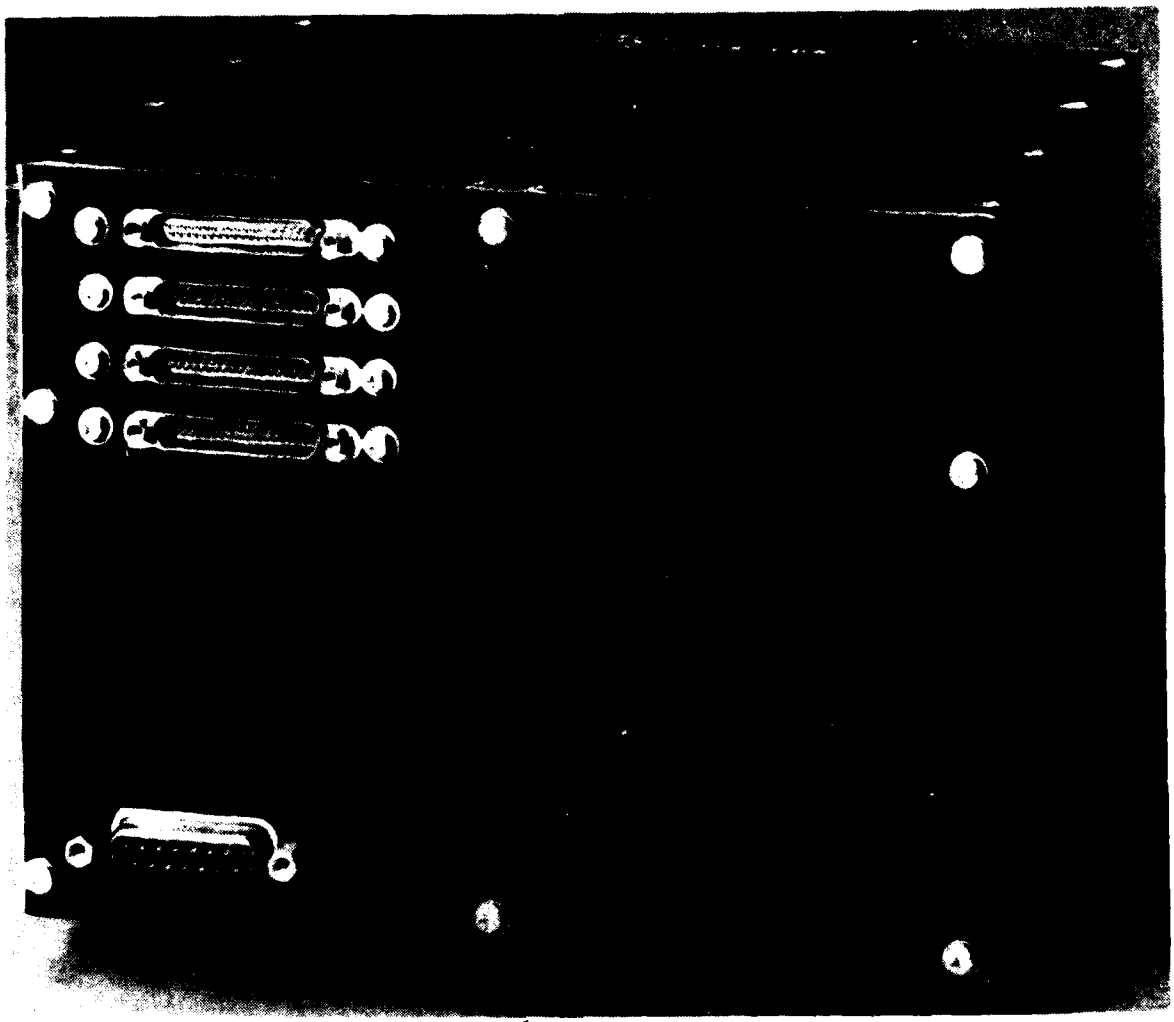


Figure 16. Federated Computer Brassboard Housing

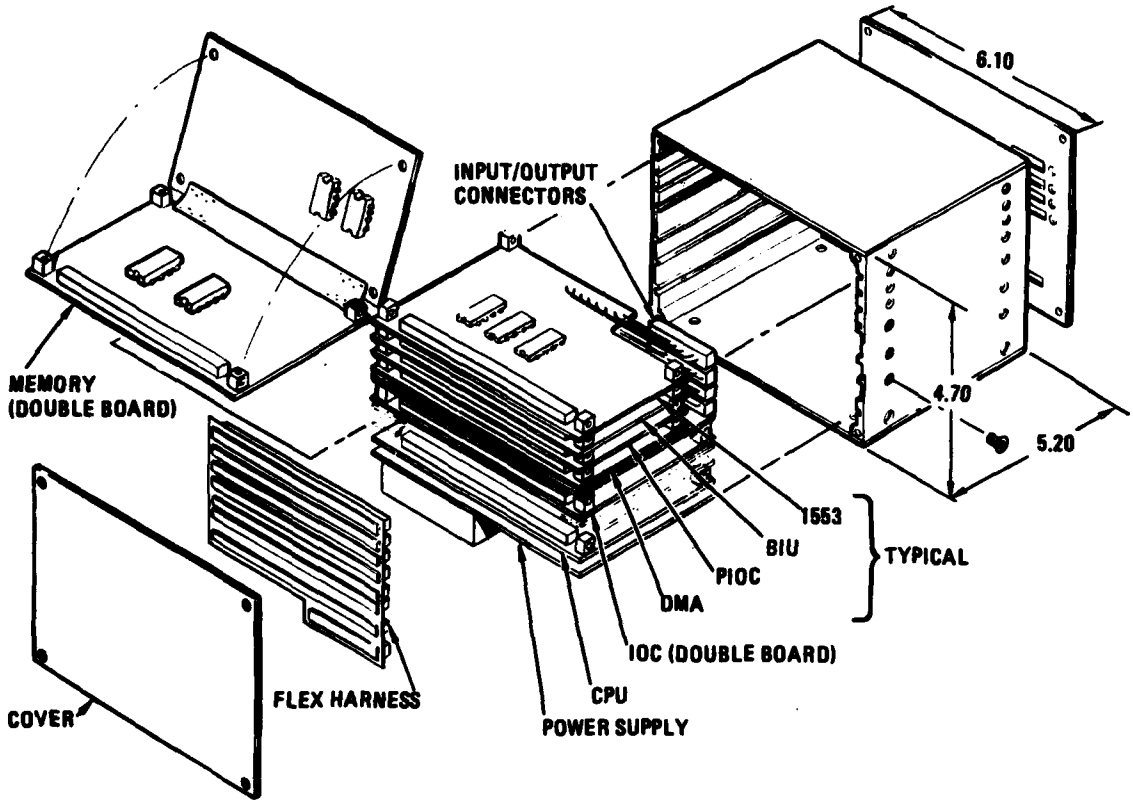


Figure 17. Federated Computer Brassboard Packaging

Figure 18. **Federated Computer Brassboard Subassemblies**

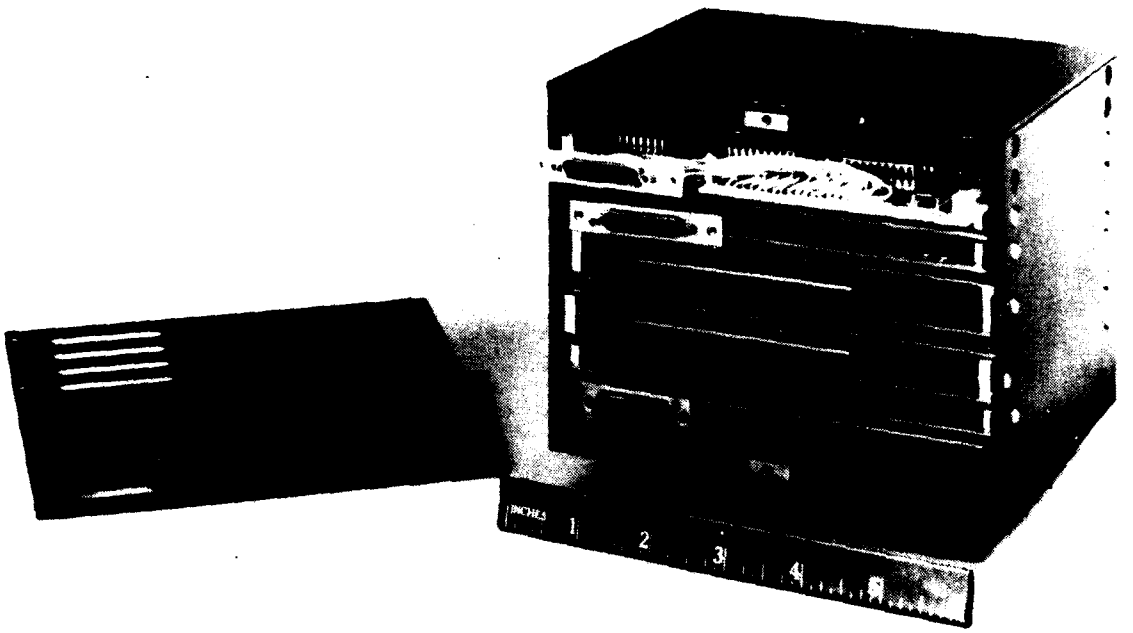


Figure 19. Federated Brassboard Computer

clock rate from 4 MHz to 6 MHz, a Class II computer design can be accomplished with very little hardware change.

The breadboard federated computers have been built, debugged, and are currently operating with little or no problems. The brassboard versions of these two computers exist today and 25 have been manufactured to meet the contract requirements.

Support test facilities developed outside the contract have performed to a high degree of confidence and have aided in accomplishing this contract's objectives. These facilities included the I/O card tester, the unibus simulator, and the RS232 card with monitor, all of which have aided in the debug and verification process.

All DIS brassboard computers have been acceptance tested per the acceptance test plan and to the procedures.

The qualification test unit brassboard for the Class I and II machines has passed the identified qualification test with minor discrepancies.

Problem Areas

Some items have been identified as potential problem areas that have either occurred during the execution of this contract, related contracts, or may pose problems if this federated computer or a similar version of this computer is put into production.

One of the main problem areas relates to the lack of second sourcing of component parts. Off-the-shelf, second-source parts obviously lead to low-cost implementations and should be used where feasible. In a few identified cases, second sourcing was not possible. Three sole-sourced parts procurement problems did arise during the execution of this contract. This included the Hitachi RAM, the Harris RAM, and AMD's 9513. The Hitachi RAM exhibited a significantly high failure rate when used at temperature extremes (25 percent), which is considered unacceptable. The Harris 6564 memory module has the drawback of not only being sole sourced, but a single chip failure on the module has the increased drawback of rejecting an entire memory module. AMD's 9513 timing controller chip was recalled by AMD for a mask problem. Quantities of the 9513 were scarce and units that operated over the entire military temperature range were difficult to find. The problems expressed here might be solved or at least alleviated as second sourcing becomes available.

Recommendations

In verifying the logic designs of the Class I and Class II federated computer hardware, it became apparent that the logical design can be upgraded, improved, and made more adaptable to future military users. This "Block II" effort should begin immediately to define the internal architecture and to evaluate what, if any, current hardware should be maintained and/or modified and to what degree. This Block II effort should include new hardware developments that have been developed by both Zilog and AMD supporting the Z8002 processor.

The new developments to be incorporated should include an evaluation of all LSI Z8000 support and peripheral chips. The extended processing unit, the virtual memory CPU and the FIFO LSI chips are a few that must be considered.

Although the DIS diagnostic station has some attributes and validity in verifying proper operation of the federated computer system, it is too expensive for a low-cost application and a more effective method must be found. This facility should not only aid in the debug of hardware but also should aid in the development of software task modules. If a low cost diagnostic development station is to be implemented, a serious question must be answered: how much of the present diagnostic station should be retained in order to achieve minimum risk and minimum cost? It should be clear that the primary goal of a low-cost workable station must not be compromised.

SECTION VIII

DISMUX

INTRODUCTION

The DISMUX bus, or weapon bus, provides the internal missile communications through a time division multiplexed, serial, round-robin passing protocol (RRPP) data transfer. The bus is composed of the several bus interface units (BIUs) interconnected via a shielded, twisted pair as shown in Figure 20.

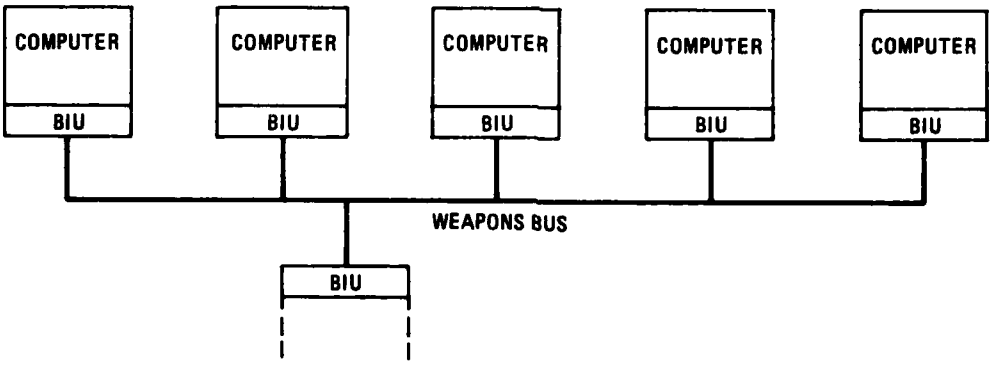


Figure 20. DISMUX Bus Provides Maximum Flexibility and Modularity

Each BIU is the required interface for a respective peripheral or computer. All BIUs are interchangeable and compatible with either class of computer. This results in optimum flexibility and internal weapon modularity.

Communication between BIUs on the DISMUX bus is in many ways similar to that between a control terminal and a remote terminal on a MIL-STD-1553B data bus. Word format, bit rate, sync, and parity are identical. Protocol is different. Furthermore, the MIL-STD-1553B bus signal voltages are attained only through the use of complex and costly line drivers (providing 6 to 12 volts peak-to-peak) while the DISMUX bus voltages can be much more economically implemented.

Since the MIL-STD-1553B bus is more structured and rigid in characteristics than the DISMUX bus, implementation of the latter can be more easily accomplished when considering the total system including software.

Each BIU complies with the following requirements:

- Interchangeability – All BIUs are identical except for firmware identification (ID) and software responses.

- Round-Robin Passing Protocol – A BIU with data ready transmits only during its turn. A BIU with no data to transmit when its turn comes transmits an End of Transmission (EOT) word so that the next BIU can take control of the bus. A BIU is programmed to sense the EOT of the previous BIU and will then transmit. In addition, a BIU receives information that it is programmed to receive every time that information appears on the DISMUX bus.
- Intermessage Interval Generation – Each BIU generates an intermessage interval (I^2), or delay time between 2 and 4 microseconds.
- Valid Word Recognition – A valid message transfer to each BIU will occur only if:
  - Each word begins with a valid sync pattern.
  - Each word contains 16 bits of data plus a parity bit, and is valid Manchester II, biphase.
  - Each word contains odd parity.
  - BIU ID and message ID fields of the beginning of message (BOM) word are correct.
  - The number of data words received is equal to the word count in the received BOM word.
- Transmitter Timer Fail Safe – The transmitter for each BIU will be enabled for transmission of messages no longer than 34 words for a maximum transmitting time of 850 microseconds ±150 microseconds.
- BIU Accesses Its Computer Via DMA – Each BIU transfers data to/from its respective computer or pheripheral via DMA. This DMA transfer is controlled by the I/O controller (includes DMAC), which decodes the Block ID and determines the starting address and the number of words to be moved.
- The BIU also has the capability of decoding monitor commands contained in the BOM message code received from an outside user element (i.e., control panels). This allows for the establishment of priority levels of interrupt within a single BIU card for testing and evaluation purposes.
- Asynchronous Operation – BIU operation is asynchronous.

In general, supervision of the DISMUX will be assigned to the DIS computer acting as the interface to the aircraft stores-management bus(s). However, the missile systems integrator is in no sense restricted to this choice since bus supervisory software or firmware will be present in all DIS computers.

The BIU used as the supervisor (with its associated computing element) satisfies the following requirements.

- Maintains RRPP – The Supervisor, through an on-card PROM, initiates an EOT to start the RRPP on power up and continuously monitors EOTs on the DISMUX bus during operation.

- Retains in memory the RRPP sequence – The supervisor, through its PROM, "knows" the BIU sequence of events on the DISMUX bus. This permits the continuous monitoring of transmissions on the bus.
- Stores the Transmitting BIU ID – The supervisor, again through its PROM, "knows" which BIU is transmitting on the DISMUX bus. If the supervisory BIU terminal does not receive any signal for 8 microseconds, it intervenes within 2 microseconds and continues the RRPP operation by transmitting the EOT word of the last identifiable BIU transmission it received. If again there is no signal for 8 microseconds, the supervisor will send the next EOT in the RRPP sequence, wait 8 microseconds, and continue in this manner until one of the BIUs responds.

The DISMUX bus interconnection will be achieved via a 77-ohm shielded, twisted pair cable.

OBJECTIVE

The objective of the DISMUX is to link up to 32 computers or terminals, in a cost effective approach. Like MIL-STD-1553B, DISMUX is a serial, one megabit per second bus; however, the purposes of the two buses are different. While the 1553B bus has been approved as the avionics standard in aircraft, the DISMUX is proposed as a more cost effective approach to internal tactical missile avionics.

Computers and terminals are connected to the DISMUX bus via a DMA type I/O card called a Bus Interface Unit (BIU). The BIU is designed to be internally equivalent to any other DMA type card yet provide all the protocol as stated in the DISMUX bus specification.

APPROACH

The BIU gives the DIS computer access to the DISMUX bus. The BIU is required to have the ability to take any place in the RRPP, to discriminate between messages on the DISMUX bus, and to be able to function as a bus monitor. These functions are selected through the use of firmware and hardware programming.

The block diagram Figure 21 shows the major logic sections on the card and the most important signal lines.

Receiver/Transmitter

The receiver/transmitter consists of a Texas Instruments 55119 driver/receiver chip, a 12-bit counter, and a flip-flop. The 55119 is a differential transistor, transistor logic (TTL) line driver/receiver selected for its small package size. It is used to drive and receive data on the DISMUX bus, a 77-ohm twisted, shielded pair cable. The counter and flip-flop are used to provide the fail-safe timer function. This function automatically disables the transmitter when the

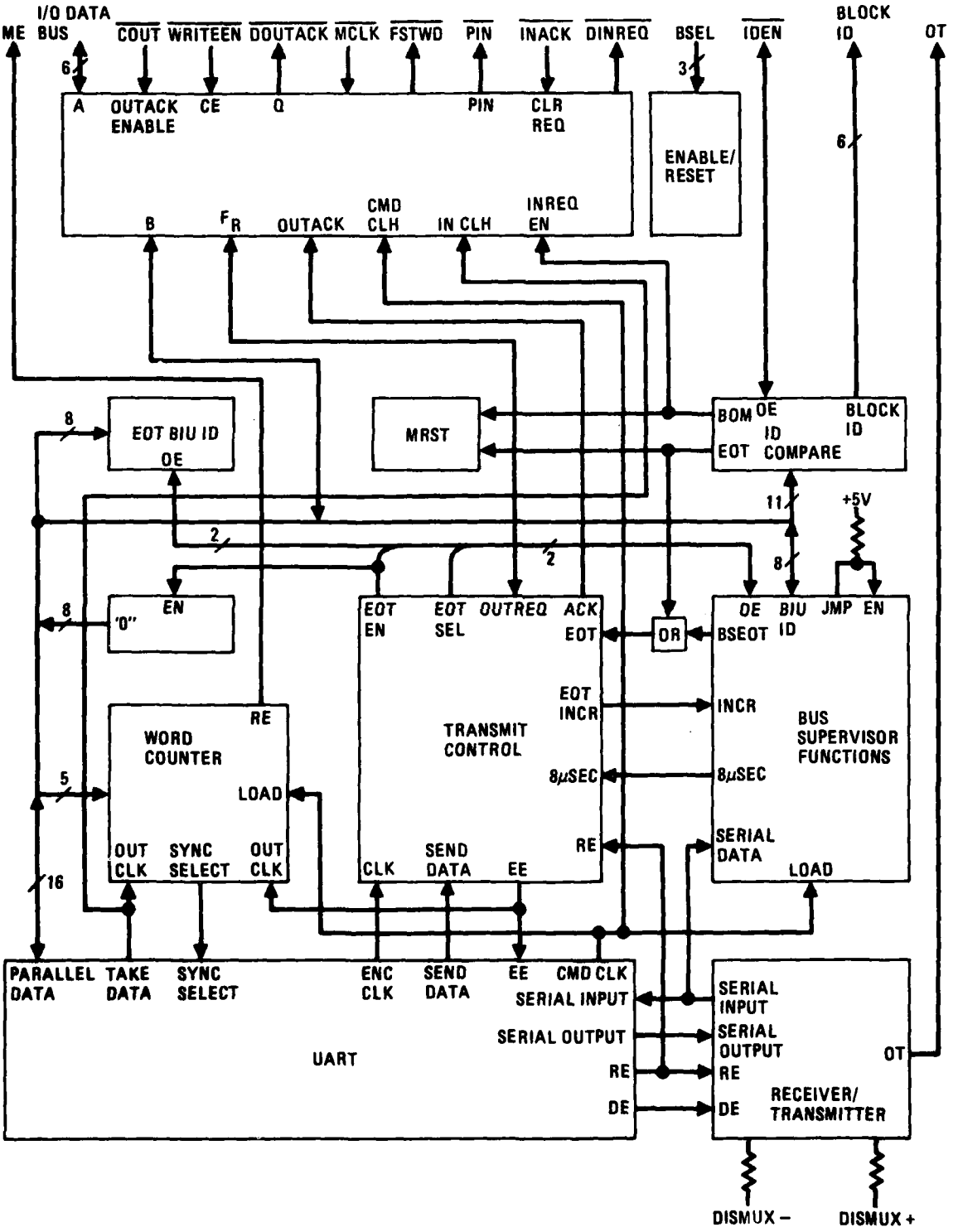


Figure 21. BIU Block Diagram

BIU attempts to make a transmission longer than the maximum allowable message length. Whenever the driver is enabled the counter is also enabled. If a count of 380_{16} (896 microseconds) is reached, the flip-flop is set, generating the overtime (OT) flag and forcing the driver enable line inactive.

The OT flag can be read by the CPU through the I/O controller as a status line. To re-enable the driver, it is necessary to reset the flip-flop and the entire BIU.

UART

The UART performs the serial-to-parallel and parallel-to-serial conversion of data, generates and decodes Manchester encoded serial data, and provides timing signals for the rest of the BIU. This UART is based on a Harris Semiconductor HD-15530 CMOS Manchester Encoder-Decoder and two 54LS299 eight bit shift/storage registers. An assortment of gates and flip-flops is also included. The HD-15530 chip does the Manchester encoding and decoding. It also generates and detects parity. The 54LS299s provide the parallel-to-serial and serial-to-parallel conversions. The random logic is used to create BIU control and status lines from the signals on the 15530 chip. The line command clock (CMD CLK) gives a rising edge whenever a valid BOM or EOT is received. This clock is used by all functions on the card that need to obtain information from command words. The line send data goes high whenever a word is being sent out on the bus. It is used as an acknowledge to the transmit control logic and as a clock for word counting during outputs. The take data line goes high whenever a word is being received from the bus. This line is used to count words being received, and it is put through a delay to be used to clock the received data to the output buffer. Receiver enable (RE) and driver enable (DE) are complementary lines that either enable the data bus receiver or driver. Encoder clock (ENC CLK) is a free running 1 MHz clock used to run the transmit control state machine. The two input lines encoder enable (ENC-EN) and sync select are used to give commands to the UART. ENC-EN starts an output sequence and sync select specifies whether the word transmitted should be preceded by a command sync or a data sync.

Transmit Control

The transmit control block takes requests from the bus supervisor and the ID compare logic and, based on the state of several status lines, runs an appropriate output routine. The three routines it can run are:

a. A normal transmission in which messages consisting of BOMs and data words are sent over the DISMUX bus and followed by an EOT. For this routine to run, the output request (OUTREQ) line must be set when the preceding BIU's EOT is detected.

b. A transmission in which only the BIU's EOT is sent. This routine is run whenever the preceding BIU's EOT is detected and OUTREQ is not set. This transmission takes place when the BIU's turn in the round-robin comes, but there is nothing to be sent.

c. A transmission in which a phantom EOT is sent. This routine is run when the bus supervisor detects 8 microseconds of bus silence. In this case the EOT to be transmitted is supplied by the bus supervisor.

The transmit control consists of a Monolithic Memories, Inc. (MMI) PAL 16R8. This device is a field programmable state machine. The state diagram of this machine is shown in Figure 22. These states are defined by three lines, an unused output called $\overline{A}$, the $\overline{EOTSEL}$ line, and the $\overline{ENC\text{-}EN}$ line. A list of the possible states and the function performed in each state is given in Table 6.

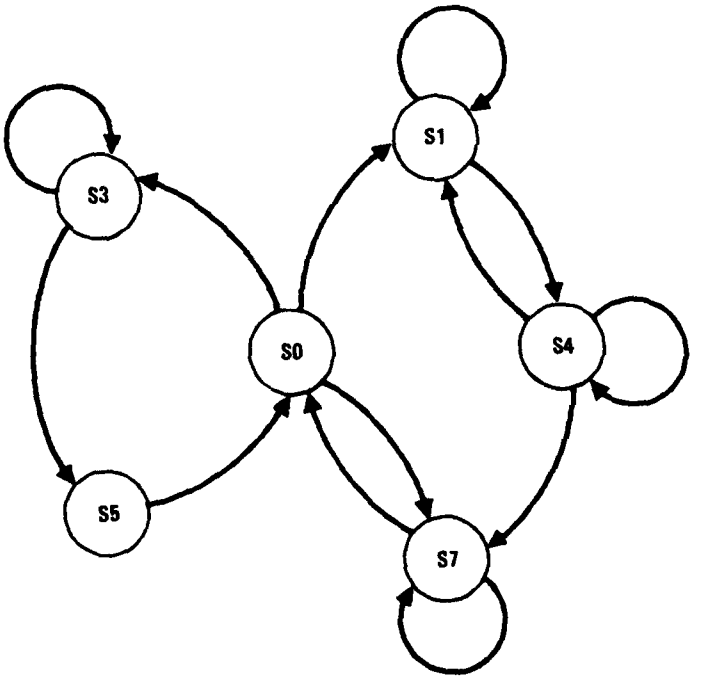


Figure 22. State Diagram of PAL16R8

On a normal transmission the machine goes from S0 to S1. It stays in S1 until the UART acknowledges the $\overline{ENC\text{-}EN}$ by taking SEND DATA high. At this point it goes to S4 and generates the ACK pulse. It waits at S4 until SEND DATA goes low indicating the word has been sent. If OUTREQ is set at this time the machine will return to S1 and send the next word. This loop from S1 to S4 will continue until the last word is sent. If OUTREQ is not set when SEND DATA goes low, the machine will change to S7, begin transmission of the EOT and increment the bus supervisor's EOT counter. After the transmission has been initiated, the machine returns to S0 and waits for the next command.

TABLE 6. BIU TRANSMISSION CONTROL STATE MACHINE

State	$\overline{A}$	$\overline{EOTSEL}$	$\overline{ENC-EN}$	Function
S0	1	1	1	Idle
S1	1	1	0	Load data from I/O port into UART and start transmission.
S3	1	0	0	Load EOT from bus supervisor into UART and start transmission.
S4	0	1	1	Wait for word to be transmitted.
S5	0	1	0	Increment bus supervisor EOT counter.
S7	0	0	0	Load BIU EOT into UART and start transmission.

When only the EOT is to be sent, the machine will move from S0 to S7. While in S7 it increments the bus supervisor's EOT counter and begins transmission of the EOT. Then, as above, once the transmission has started the machine returns to S0.

When the bus supervisor requests transmission of a phantom EOT, the machine proceeds to S3 and begins transmission of the EOT given by the bus supervisor. When the transmission begins it moves to S5 and increments the EOT counter. The machine returns to S0 from S5 regardless of input.

Although not shown in the state diagram, a low level on the $\overline{MRST}$ line will cause the transfer from any state to S0.

The equations that define this state machine and are programmed into the PAL16R8 are shown in Table 7.

Word Counter

The word counter counts words for both inputs and outputs. During inputs it counts the words between successive command words to verify that the word count field in the BOM matches the number of words actually received. During outputs it counts data words following a BOM so that it knows when to expect the next BOM and can generate a command sync for it.

The counter is based on a RCA CD40103B, complementary metal-oxide silicon (CMOS), eight-stage, presettable, synchronous down counter. The most significant two bits of the counter are not used and are tied to zeros. The least significant five bits are provided by the word count field in the BOM. Since a word count field of all zeros must be decoded as 32, the five bits are NOR'd to give a one in the sixth bit when they are all zero.

TABLE 7. TRANSMIT CONTROL STATE EQUATIONS

A=:	/A*/EOT-SEL*/ENC-EN*/IO-REQ*/SEND-DATA*BSEOT*/MRST+
	/A*/EOT-SEL*ENC-EN*SEND-DATA*/MRST+
	/A*EOT-SEL*ENC-EN*SEND-DATA*/MRST+
	A*/EOT-SEL*/ENC-EN*SEND-DATA*/MRST+
	A*/EOT-SEL*/ENC-EN*/IO-REQ*/SEND-DATA*/MRST+
	A*EOT-SEL*ENC-EN*/SEND-DATA*/MRST
EOT-SEL=:	/A*/EOT-SEL*/ENC-EN*8USEC*REC-EN*/BSEOT*/MRST+
	/A*/EOT-SEL*/ENC-EN*/IO-REQ*/SEND-DATA*BSEOT*/MRST+
	/A*EOT-SEL*ENC-EN*/SEND-DATA*/MRST+
	A*/EOT-SEL*/ENC-EN*/IO-REQ*/SEND-DATA*/MRST+
	A*EOT-SEL*ENC-EN*/SEND-DATA*/MRST
ENC-EN=:	/A*/EOT-SEL*/ENC-EN*/SEND-DATA*BSEOT*/MRST+
	/A*/EOT-SEL*/ENC-EN*8USEC*REC-EN*/BSEOT*/MRST+
	/A*/EOT-SEL*ENC-EN*/SEND-DATA*/MRST+
	/A*EOT-SEL*ENC-EN*/MRST+
	A*/EOT-SEL*/ENC-EN*/SEND-DATA*/MRST+
	A*EOT-SEL*ENC-EN*/SEND-DATA*/MRST
BUS-SEL=:	/A*/EOT-SEL*/ENC-EN*8USEC*REC-EN*/BSEOT*/MRST+
	/A*EOT-SEL*ENC-EN*/SEND-DATA*/MRST
	/EOT-INC=:/EOT-SEL+/ENC-EN+/SEND-DATA+MRST
	/ACK:A+EOT-SEL+/ENC-EN+/SEND-DATA+MRST

The message error (ME) flag is set if either one of the data words in a received message has bad parity or if the count is not at zero at the completion of a received message. Messages the BIU is not programmed to receive do not affect the ME flag.

"0"

The "0" block is a 54LS244, octal, tri-state line driver. All of the inputs are tied to ground. This device drives the least significant byte of the data bus to zeros when an EOT is loaded into the UART.

OR

The OR function is provided by part of a 54LS399 quad 2-input multiplexer with storage. One input of this multiplexer is tied to the ID compare EOT line. The other input is tied to the bus supervisor $\overline{\text{BSEOT}}$ line. This is done to allow the BIU's turn in the round robin to be detected by either the ID compare function or the bus supervisor. In normal operation the ID compare detects when the BIU is to transmit from a received EOT. When the BIU is acting as a bus supervisor and sends the EOT for the BIU it is to follow instead of receiving it, the bus supervisor determines that it's round robin turn has arrived.

$\overline{\text{MRST}}$

It is possible for a computer to receive a hardware reset in a DISMUX command word. This function is provided by the ID compare logic and the $\overline{\text{MRST}}$ block. It is implemented by programming a location in the ID compare EPROM to have ones in both the EOT and BOM bits. When the command word corresponding to the address of this location is received, the $\overline{\text{MRST}}$ line of the DIS computer will go low for approximately 15 microseconds. This will reset all of the BIU logic as well as the rest of the computer in the same manner as a power on reset. The command word will not be sent into memory since the BIU and DMA controller are reset as soon as it can be recognized.

$\overline{\text{MRST}}$ is an open collector line in the DIS computer backplane that can be driven by either the BIU, the power supply, or the CPU.

Enable/Reset

The CPU can reset and disable the BIU by writing zeros to the three BSEL lines associated with the card slot the BIU occupies. If any of these lines is a one, the BIU is enabled and operates normally. This function is provided through the use of half of a 54LS139 decoder. When the A, B, and $\overline{\text{G}}$ inputs are all low, the reset line on the BIU will be set low and remain low until one of them is set to a one.

EOT BIU ID

A BIU can be programmed to take any place in the DISMUX round robin passing protocol (RRPP). The ID on the EOT sent out by a BIU is programmed by five jumpers on the card. These jumpers control the state of the inputs to a 54LS244 tri-state driver. This output of this driver is enabled when the BIU's EOT is loaded into the UART.

Figure 23 shows the assignments of the EOT ID jumpers on the BIU card. When a jumper is in place, the corresponding bit in the EOT is a zero. When it is removed, the bit becomes a one.

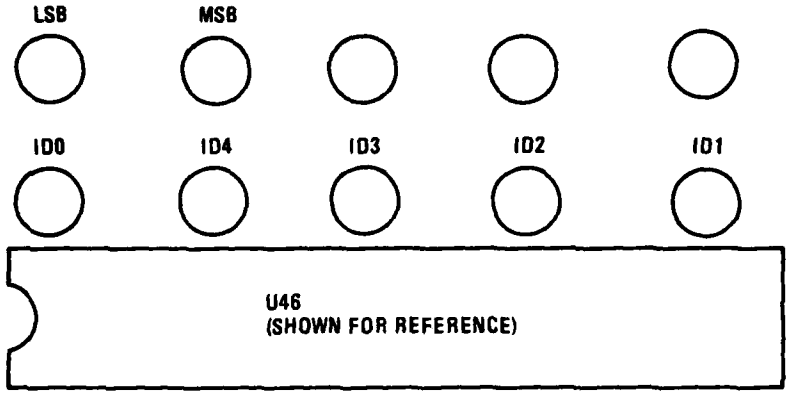


Figure 23. Assignment of EOT ID Jumpers on BIU Card

Interface Logic

The BIU moves DISMUX messages in and out of memory via one of the four DMA channels on the I/O controller. The request and status lines required by the DIS DMA controller are generated within the block titled interface logic. This block contains two AM 2950 eight-bit bidirectional I/O ports, several flip-flops for handshake and status generation, and the gates required to make them work.

The following signals are used while the BIU is receiving data from the DISMUX bus and sending it into memory.

$\overline{\text{PIN}}$ is an active low line used by the DMA controller to tell that the BIU has an input in progress. This line is generated by a J-K flip-flop clocked whenever a command word is received. The BOM bit from the ID compare EPROM is connected to the K input while the control bit of the received word is tied to the J input. Thus reception of a BOM to be received sets $\overline{\text{PIN}}$ low until an EOT is received, which will set it back high. The falling edge of $\overline{\text{PIN}}$ clocks a D flip-flop to generate the first word ($\overline{\text{FSTWD}}$) flag. When the word is read, this flag is cleared by input acknowledge (INACK) and remains inactive until the beginning of the next input. $\overline{\text{INACK}}$ is also tied to the AM 2950 output enable and clear line. The low level allows the BIU's received word to be enabled onto the data bus and the rising edge clears the flag that generates DMA input request ($\overline{\text{DINREQ}}$). $\overline{\text{DINREQ}}$ is simply the inversion of the S flag on the AM 2950 that is set whenever a word is loaded into the S register. Control of which words are loaded into the S register is provided by a flip-flop operating in a manner similar to the $\overline{\text{PIN}}$ flip-flop. The major difference is that this signal is active only for messages to be received. This allows the BIU to pick only the messages it is programmed to receive out of transmissions containing more than one BOM. The logic around this flip-flop is designed to store in memory EOTs following received messages while ignoring all others.

A second set of lines is used on outputs. When the DMA controller begins an output the computer output (COUT) line will go low. This line is "AND'd" with the write enable line (WRITEEN) and the MRST line to clear the DMA output acknowledge ($\overline{\text{DOUTACK}}$). This serves to disable the $\overline{\text{DOUTACK}}$ when the last word is loaded onto the BIU. $\overline{\text{DOUTACK}}$ is generated from the transmit control's ACK line. This occurs as the word to be transmitted is loaded from the R register in the AM 2950s into the UART. When the DMA controller receives a $\overline{\text{DOUTACK}}$ it can move another word from memory to the BIU. Words are written into the AM 2950 R registers by the use of a clock enable line ($\overline{CE}_R$) and a rising edge on the clock input (CP_R). $\overline{\text{WRITEEN}}$ is used as the $\overline{CE}_R$ line and $\overline{\text{MCLK}}$, a clock that is synchronized with CPU internal cycles, is inverted to become the CP_R line. Loading data into the AM 2950 sets the F_R flag, which goes to the transmit control logic to indicate that a word is ready to be sent. It is not necessary for the BIU's turn in the RRPP to occur before the DMA controller can begin an output. In normal operation the first BOM will be loaded onto the BIU and then be held there until it can be sent, after which a new word will be requested approximately every 20 microseconds until the transmission is complete.

As a summary, Table 8 gives the name and function of all BIU signal lines at the computer backplane.

Bus Supervisor Functions

One of the BIUs on a DISMUX bus must have its bus supervisor functions enabled. The bus supervisor initiates the round robin passing protocol (RRPP) at system power on and then maintains the RRPP when one or more of the BIUs on the bus is not transmitting its EOT correctly.

The bus supervisor is made up of a round robin table stored in a 2716 EPROM, two 54LS399 multiplexed input latches, a 54LS244 line driver, a 54LS161 four bit counter and a D flip-flop. The round robin table is stored in the upper 64 locations in the EPROM. Even though most of the locations are unused, this 2K × 8 EPROM was chosen due to its advantages of being a multiple-sourced, industry-standard device requiring a single voltage supply. The feature of erasability is required due to the developmental nature of DIS and related programs in which frequent configuration changes will take place. Table 9 shows how to program this EPROM to make the bus supervisor work correctly. The $\overline{\text{BSEOT}}$ bit, 05, is a zero at the address containing the bus supervisor's EOT. Note that 05 must always be a one in the first half of the table and can only be active in the second half of the table. A location in the table associated with a BIU not in the system should be filled with the ID of a default BIU within the system so that if an illegal ID appears on the bus, the supervisor will be able to restart the bus.

The 54LS399 latches hold the BIU ID that will be sent out the next time an 8-microsecond gap is detected on the bus. This ID also makes up part of the EPROM address so that the output of the EPROM will be the ID of the next BIU on the bus. During normal operation these latches are loaded from any command word received by the BIU. This causes the supervisor to restart the bus at the

TABLE 8. BIU PIN DESCRIPTION

Name	I/O	Description
OT	OUT	Overtime flag equal to 896 microseconds; disables transmitter; resets on master reset.
ME	OUT	Message error flag; active high for error detected on any received word or number of words.
$\overline{\text{COUT}}$	IN	Active low during entire output transfer.
I/O BUS	I/O	16-bit bidirectional parallel bus.
$\overline{\text{INACK}}$	IN	Active low from I/O controller acknowledging input data.
$\overline{\text{DINREQ}}$	OUT	Active low from BIU requesting service from I/O controller.
$\overline{\text{WRITEEN}}$	IN	Active low from I/O controller enabling loading of BIU output data on MCLK edge.
BLOCK ID	OUT	6-bit encoded lines from BIU card to I/O controller equivalent to starting address.
$\overline{\text{FSTWD}}$	OUT	Active low from BIU card signalling controller that first word of transfer is in progress.
BSEL 1-3	IN	If all lines equal to zero; reset function (common to all I/O cards).
$\overline{\text{MRST}}$	OUT	Master reset, active low.
$\overline{\text{DOUTACK}}$	OUT	Active low from BIU acknowledging previous transmitted word from I/O controller and indicating ready for next word.
$\overline{\text{MCLK}}$	IN	Processor CLK used in I/O exchange.
$\overline{\text{PIN}}$	OUT	Active low during entire transfer of received bus data to controller.
$\overline{\text{IDEN}}$	IN	Active low signal from controller enabling the block ID on the tri-state controller ID bus.
$\overline{\text{MCLKRTN}}$	IN	Return line for MCLK.
CARD ID	OUT	Two lines indicating the I/O card type; for the BIU both lines are tied low.

point in the round robin where it stopped when an 8-microsecond gap is detected. During transmission of an EOT, the latches are loaded from 04 through 00 of the EPROM. This increments to the ID of the next BIU in the round robin in case another 8 microseconds passes with no bus activity.

TABLE 9. BUS SUPERVISOR EPROM

$Address_{16}$	06*	05-01
7C0	1	BIU ID TO FOLLOW BIU 0
7C1	1	BIU ID TO FOLLOW BIU 1
7C2	1	BIU ID TO FOLLOW BIU 2
7C3	1	BIU ID TO FOLLOW BIU 3
7C4	1	BIU ID TO FOLLOW BIU 4
7C5	1	BIU ID TO FOLLOW BIU 5
7C6	1	BIU ID TO FOLLOW BIU 6
7C7	1	BIU ID TO FOLLOW BIU 7
7C8	1	BIU ID TO FOLLOW BIU 8
7C9	1	BIU ID TO FOLLOW BIU 9
7CA	1	BIU ID TO FOLLOW BIU 10
7CB	1	BIU ID TO FOLLOW BIU 11
7CC	1	BIU ID TO FOLLOW BIU 12
7CD	1	BIU ID TO FOLLOW BIU 13
7CE	1	BIU ID TO FOLLOW BIU 14
7CF	1	BIU ID TO FOLLOW BIU 15
7D0	1	BIU ID TO FOLLOW BIU 16
7D1	1	BIU ID TO FOLLOW BIU 17
7D2	1	BIU ID TO FOLLOW BIU 18
7D3	1	BIU ID TO FOLLOW BIU 19
7D4	1	BIU ID TO FOLLOW BIU 20
7D5	1	BIU ID TO FOLLOW BIU 21
7D6	1	BIU ID TO FOLLOW BIU 22
7D7	1	BIU ID TO FOLLOW BIU 23
7D8	1	BIU ID TO FOLLOW BIU 24
7D9	1	BIU ID TO FOLLOW BIU 25
7DA	1	BIU ID TO FOLLOW BIU 26
7DB	1	BIU ID TO FOLLOW BIU 27

*08 and 07 are not used and are therefore "don't care."

TABLE 9. BUS SUPERVISOR EPROM (CONTINUED)

$Address_{16}$	06*	05-01
7DC	1	BIU ID TO FOLLOW BIU 28
7DD	1	BIU ID TO FOLLOW BIU 29
7DE	1	BIU ID TO FOLLOW BIU 30
7DF	1	BIU ID TO FOLLOW BIU 31
7E0	BSEOT	BIU ID TO FOLLOW BIU 0
7E1	BSEOT	BIU ID TO FOLLOW BIU 1
7E2	BSEOT	BIU ID TO FOLLOW BIU 2
7E3	BSEOT	BIU ID TO FOLLOW BIU 3
7E4	BSEOT	BIU ID TO FOLLOW BIU 4
7E5	BSEOT	BIU ID TO FOLLOW BIU 5
7E6	BSEOT	BIU ID TO FOLLOW BIU 6
7E7	BSEOT	BIU ID TO FOLLOW BIU 7
7E8	BSEOT	BIU ID TO FOLLOW BIU 8
7E9	BSEOT	BIU ID TO FOLLOW BIU 9
7EA	BSEOT	BIU ID TO FOLLOW BIU 10
7EB	BSEOT	BIU ID TO FOLLOW BIU 11
7EC	BSEOT	BIU ID TO FOLLOW BIU 12
7ED	BSEOT	BIU ID TO FOLLOW BIU 13
7EE	BSEOT	BIU ID TO FOLLOW BIU 14
7EF	BSEOT	BIU ID TO FOLLOW BIU 15
7F0	BSEOT	BIU ID TO FOLLOW BIU 16
7F1	BSEOT	BIU ID TO FOLLOW BIU 17
7F2	BSEOT	BIU ID TO FOLLOW BIU 18
7F3	BSEOT	BIU ID TO FOLLOW BIU 19
7F4	BSEOT	BIU ID TO FOLLOW BIU 20
7F5	BSEOT	BIU ID TO FOLLOW BIU 21
7F6	BSEOT	BIU ID TO FOLLOW BIU 22
7F7	BSEOT	BIU ID TO FOLLOW BIU 23

TABLE 9. BUS SUPERVISOR EPROM (CONCLUDED)

$Address_{16}$	06*	05-01
7F8	BSEOT	BIU ID TO FOLLOW BIU 24
7F9	BSEOT	BIU ID TO FOLLOW BIU 25
7FA	BSEOT	BIU ID TO FOLLOW BIU 26
7FB	BSEOT	BIU ID TO FOLLOW BIU 27
7FC	BSEOT	BIU ID TO FOLLOW BIU 28
7FD	BSEOT	BIU ID TO FOLLOW BIU 29
7FE	BSEOT	BIU ID TO FOLLOW BIU 30
7FF	BSEOT	BIU ID TO FOLLOW BIU 31

The 54LS244 drives the upper eight bits of the data bus when the bus supervisor has an EOT loaded into the UART. The most significant five inputs are supplied by the ID stores in the 54LS399s. The next most significant bit is tied to Vcc to put a one in the control bit of the EOT. The last two bits are tied to ground to put zeros into the unused bits of the EOT as required by the DIS specification.

The counter and D flip-flop are used to detect 8-microsecond gaps on the DISMUX bus. The D flip-flop is used as an edge detector for the Manchester data appearing on the bus. This is done by clocking the flip-flop on a 2-MHz clock and comparing the output with the input. The output is different from the input when an edge has occurred. The output of this edge detector is used as the asynchronous clear on the four bit counter clocked at 1 MHz. The output line QD is used to indicate when 8 microseconds have passed without detection of an edge. To prevent a non-bus supervisor from generating the 8-microsecond signal to the transmit control logic, the output of the edge detector is disconnected from the counter leaving the clear line always active. The receiver enable line is tied to the LOAD input to reinitialize and disable the counter while the BIU is transmitting.

ID Compare

The ID compare function is performed by a 2716 EPROM and a 54LS374 octal tri-state latch. The 11 address bits of the EPROM are tied to the most significant bits of the data bus. This gives a unique location in the EPROM for each possible DISMUX command word. Inside the EPROM is stored information used to discriminate between messages on the bus to be received or ignored, the block ID to assign to each incoming message, the bit used to inform the BIU that the EOT it follows has just been received, and the data required to decode a computer reset ($\overline{\text{MRST}}$) from an incoming message. The bit definitions of the output of this EPROM are shown in Figure 24.

07	06	05	04	03	02	01	00
EOT	BOM	BLK 6	BLK 5	BLK 4	BLK 3	BLK 2	BLK 1

EOT = 1 AT ADDRESS CORRESPONDING TO PRECEDING BIUs EOT
= 0 ALL OTHER ADDRESSES*

BOM = 1 AT ADDRESSES CORRESPONDING TO BOM OF MESSAGE TO BE RECEIVED
= 0 ALL OTHER ADDRESSES*

BLK6-BLK1 = BLOCK ID AT ADDRESSES CORRESPONDING TO BOMs TO BE RECEIVED
= DON'T CARE ELSEWHERE

*PROGRAMMING ONES IN BOTH THE EOT AND BOM BITS ALLOWS COMMAND WORDS FROM THE DISMUX TO BE DECODED INTO MRST.

Figure 24. ID Compare EPROM Output

The block ID of the first BOM received in a transmission is stored in the 54LS374 register. This saves the block ID to be loaded with the message error bit into the I/O controller at the end of an input. The output enable of this register is tied to $\overline{\text{IDEN}}$ from the I/O controller.

Programmable Logic

To save space and power on the BIU, two of MMI's programmable array logic (PAL) devices are used. These are U31, a PAL12H6, and U32, a PAL16R8. The 12H6 contains user programmable AND-OR links while the 16R8 provides AND-OR and flip-flop elements. Figures 25 and 26 should be referred to when trying to relate the logic equations to the hardware. Tables 10 and 11 give the equations that define their operation. More data on the PAL16R8 appears in a preceding section covering transmit control.

RESULTS

The DISMUX bus has been implemented and tested with great success. The BIU flight card is compatible with other DMA type cards (see Figure 27, parts lay-out). The supervisor has been able to maintain bus communications both at power up and during simulated failures of other computers. Common mode noise does not seem to be a problem.

CONCLUSION/RECOMMENDATIONS

The BIU, although functionally working and successful in the DIS system, could be improved. The following lists some proposed improvements if the DISMUX were to be incorporated into a production system:

a. Reduce parts count. Since the design is basically 2 years old and is implemented with that technology, a relatively large number of parts were

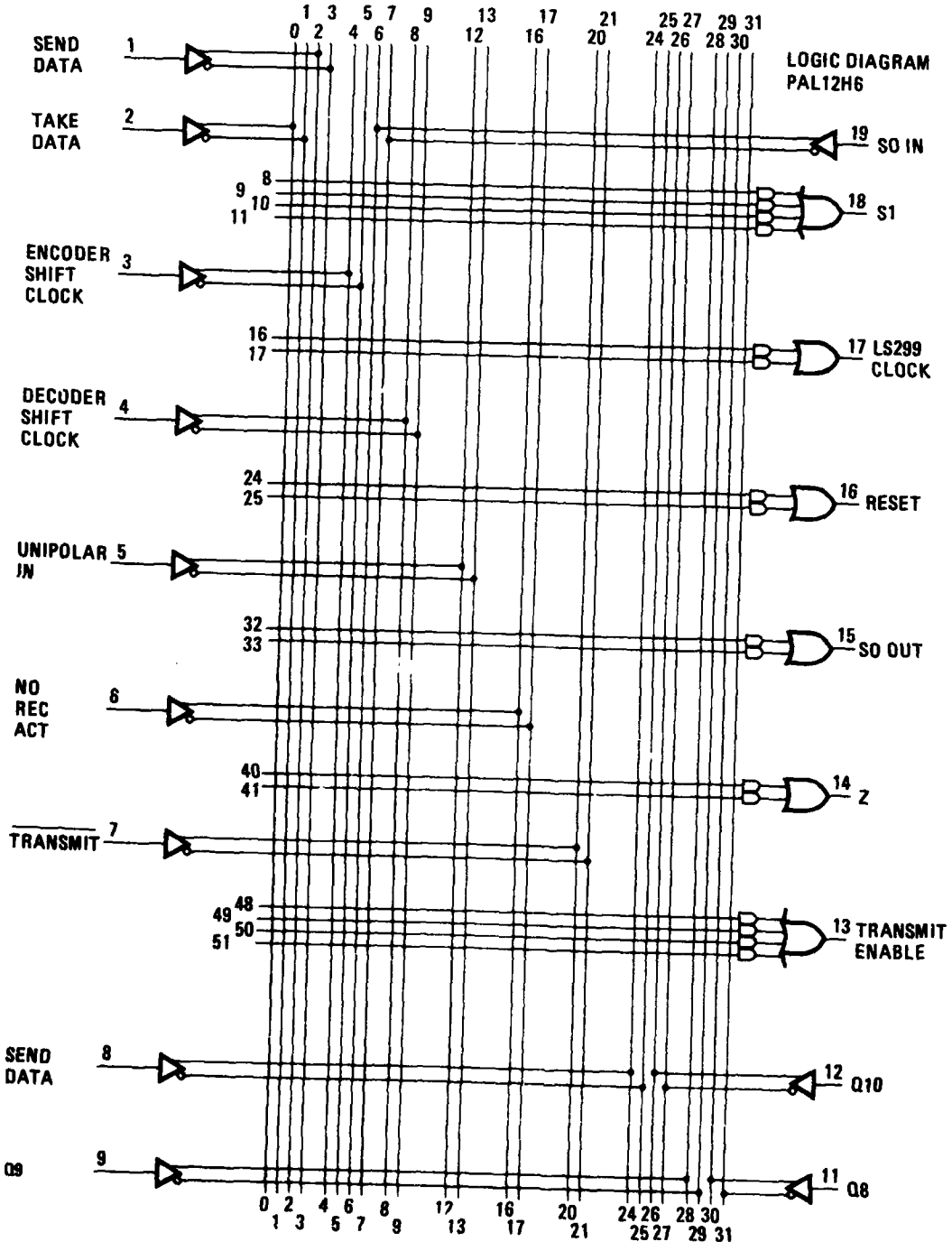


Figure 25. PAL12H6 Array

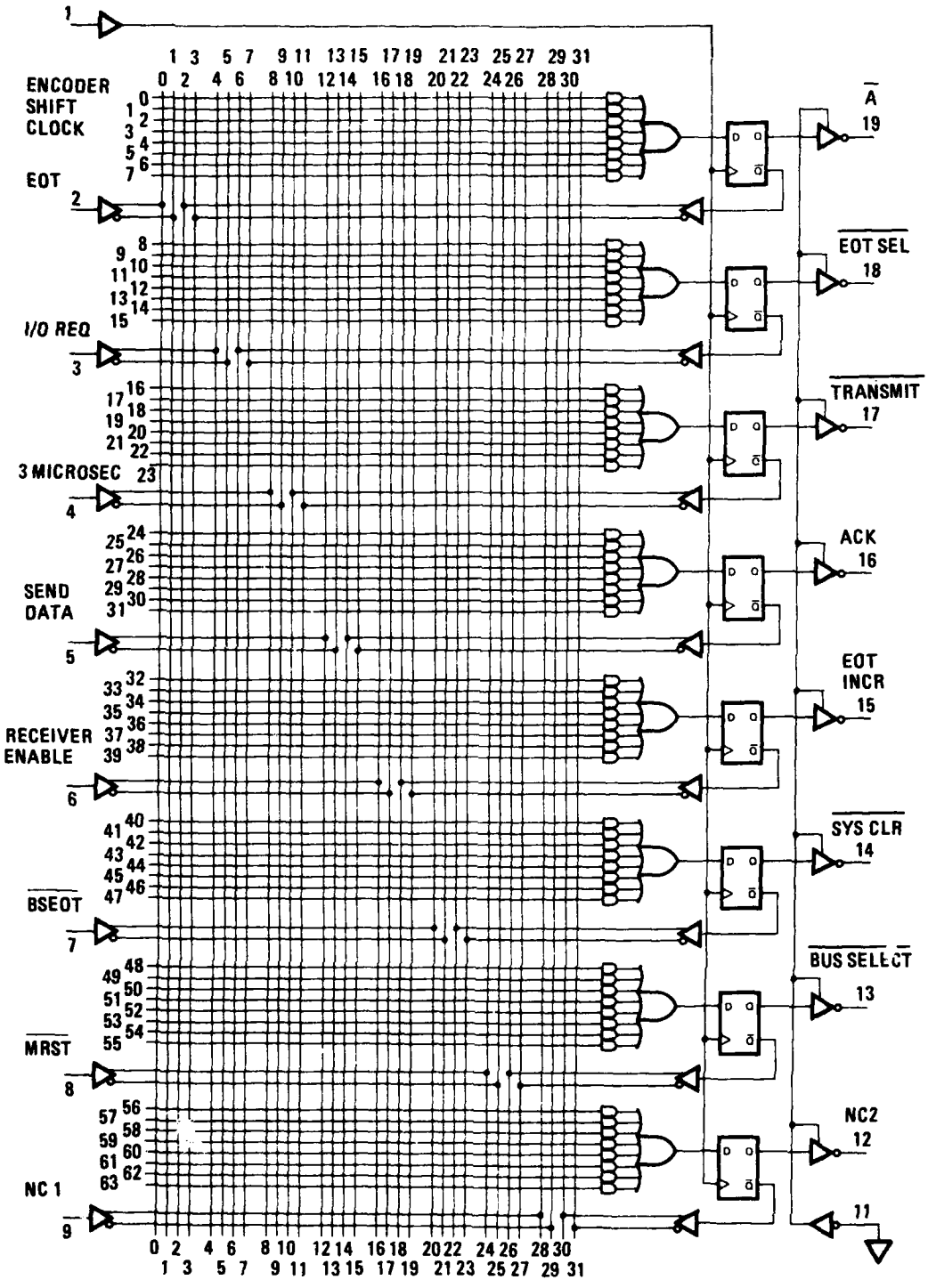


Figure 26. PAL16R8 Array

TABLE 10. PAL12H6 U31 LOGIC EQUATIONS

SOP=/SD * EE

TE= EE + SD + SDT3 + SOIN

Z= 8 * 9 * Q10

LSC= SD * /EC + TD * DC

CR= UI * /WR + /UI * WR

S1= SOIN + SD + TD

necessary to implement the function, which forced the function to be implemented with flatpacks. By using the current state-of-the-art technology, the parts count can be drastically reduced, thereby eliminating the flatpacks and reducing the total cost.

b. The bus supervisor uses a 2K × 8 PROM for storing the sequence of up to 32 possible terminals. This waste of memory space, cost, size, and power of the large PROM part is not necessary. The only reason to keep this 2716, 2K × 8 PROM was the fact that it is ultraviolet (UV) erasable, which is desirable in a new technology or research and development (R&D) effort. However, in production, where the sequence has been established, a cheaper 32 × 8 PROM should be used.

c. If the card were redesigned with current large scale integration (LSI) devices reducing parts count and required board area, there might be adequate space to provide double buffering of messages on card. This would aid in preventing any overwrite problems or interrupt service timing problems.

d. The present computer allows only one pending input interrupt and one pending output interrupt per card slot. It is, therefore, possible in an extreme case to lose a short BIU message because there is no means of stacking interrupts. If the BIU could stack pending interrupts loss of messages would probably be impossible, although this may also require some I/O controller changes.

In conclusion, the DISMUX bus has been the subject of scrutiny by bus standardization people. It should be noted as a final observation, that DISMUX offers some merits over the 1553B for low cost tactical missile applications. This does not suggest that the DISMUX bus should be considered as a standard, but that system applications should determine the most appropriate choice.

TABLE 11. TRANSMIT CONTROL STATE EQUATIONS

A=:	/A*/EOT-SEL*/ENC-EN*/IO-REQ*/SEND-DATA*BSEOT*/MRST+
	/A*/EOT-SEL*ENC-EN*SEND-DATA*/MRST+
	/A*EOT-SEL*ENC-EN*SEND-DATA*/MRST+
	A*/EOT-SEL*/ENC-EN*SEND-DATA*/MRST+
	A*/EOT-SEL*/ENC-EN*/IO-REQ*/SEND-DATA*/MRST+
	A*EOT-SEL*ENC-EN*/SEND-DATA*/MRST
EOT-SEL=:	/A*/EOT-SEL*/ENC-EN*8USEC*REC-EN*/BSEOT*/MRST+
	/A*/EOT-SEL*/ENC-EN*/IO-REQ*/SEND-DATA*BSEOT*/MRST+
	/A*EOT-SEL*ENC-EN*/SEND-DATA*/MRST+
	A*/EOT-SEL*/ENC-EN*/IO-REQ*/SEND-DATA*/MRST+
	A*EOT-SEL*ENC-EN*/SEND-DATA*/MRST
ENC-EN=:	/A*/EOT-SEL*/ENC-EN*/SEND-DATA*BSEOT*/MRST+
	/A*/EOT-SEL*/ENC-EN*8USEC*REC-EN*/BSEOT*/MRST+
	/A*/EOT-SEL*ENC-EN*/SEND-DATA*/MRST+
	/A*EOT-SEL*ENC-EN*/MRST+
	A*/EOT-SEL*/ENC-EN*/SEND-DATA*/MRST+
	A*EOT-SEL*ENC-EN*/SEND-DATA*/MRST
BUS-SEL=:	/A*/EOT-SEL*/ENC-EN*8USEC*REC-EN*/BSEOT*/MRST+
	/A*EOT-SEL*ENC-EN*/SEND-DATA*/MRST
	/EOT-INC=:/EOT-SEL+/ENC-EN+/SEND-DATA+MRST
	/ACK=:A+EOT-SEL+/ENC-EN+/SEND-DATA+MRST

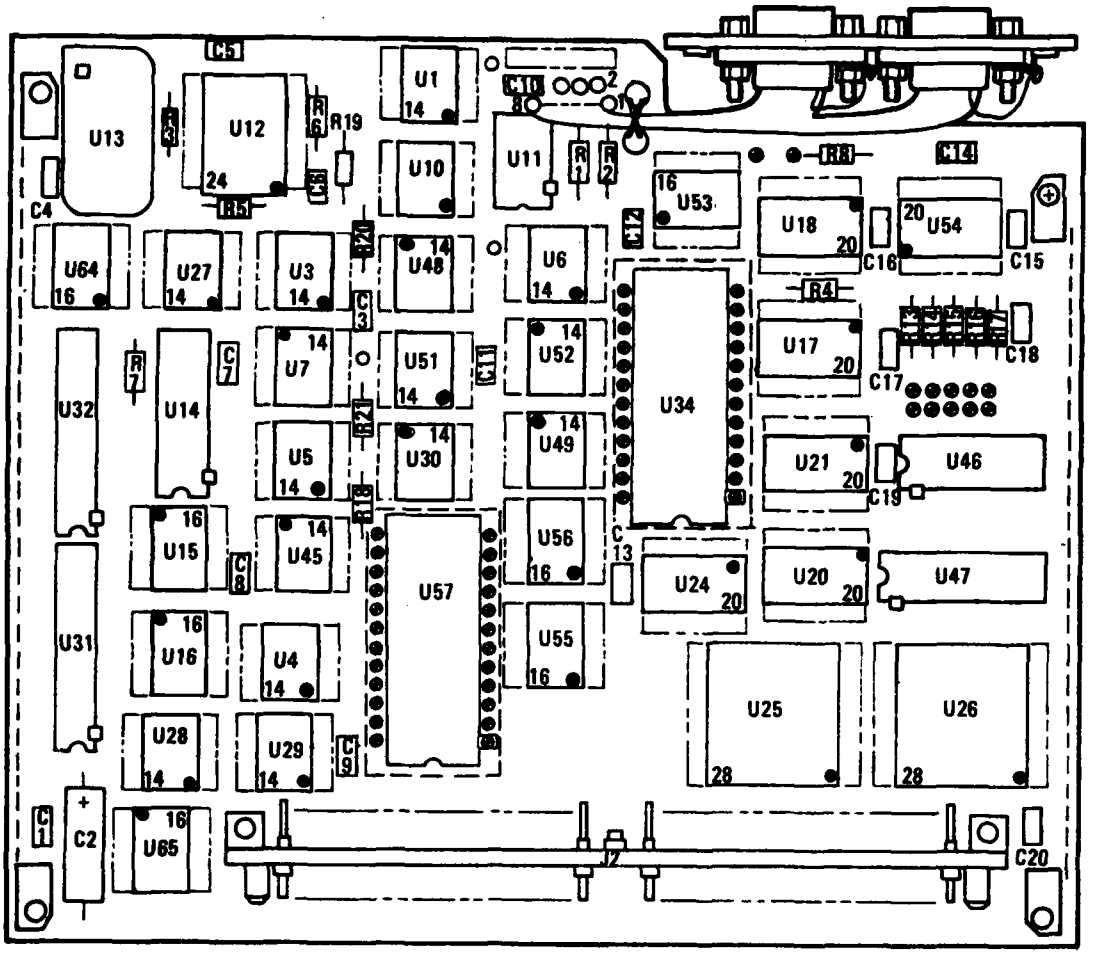


Figure 27. BIU Layout

SECTION IX

STANDARD INTERFACES

INTRODUCTION

Each of the computers connects to its respective peripheral (including the DISMUX bus) through from one to four of the following five types of interfaces:

- Serial-programmed handshake
- Parallel-programmed handshake
- Parallel DMA
- Serial MIL-STD-1553B
- BIU (the interface required for DISMUX bus communication)

Since computer-to-DISMUX bus transfers are always accomplished via the BIU card, only three I/O slots remain for any combination of the other four types of I/O cards. Figure 28 shows our proposed computer I/O configuration.

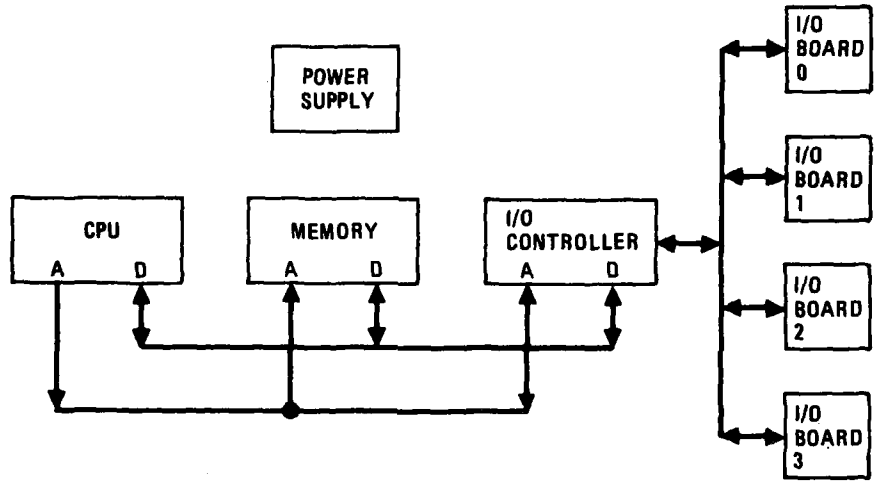


Figure 28. I/O Configuration

The I/O controller performs the functions of priority encoding, controlling I/O between the card slots and the computer, status storage, and temporary data storage. This controller consists of a multichannel DMA controller (DMAC) plus temporary registers and interrupt control logic.

Maximum interface flexibility has been obtained through an approach using card slot position to determine I/O priority rather than by building priority encoders into each interface card. This allows the use of two or more identical interface cards by the same computer without requiring additional priority encoding hardware to distinguish between them.

OBJECTIVE

The objective of the I/O card development (other than to link to peripheral devices) was to establish a selection of I/O standards that could be used for practically any peripheral requirement. A Draper study determined the types of I/O implemented under the DIS contract. However, since their report (Digital Processing Analyses/Partitioning – DPAP Final Report), Draper has modified its position on standard I/Os. The external requirements are obviously still dependent upon the peripheral, but all I/Os internally should be handled by the computer as DMA.

A contract objective in implementing these designs was to package all I/O types on the same physical size board to facilitate "mixing and matching" I/Os. In addition, circuitry in the I/O controller had to be included that could determine the card type, handshake data in either direction to the CPU or memory, prioritize I/O interrupts, and hold status information pertinent to the I/O exchange.

APPROACH

The approach taken with each card will be discussed in the following subsections.

Serial I/O

The serial input/output control (SIOC) card was designed to act as a variable bit-rate interface between the computer and a peripheral device. Programmed handshake was chosen as the type of protocol, rather than direct-memory-access, so that the CPU card could do real-time data manipulation without having to access data from memory.

The main design goals for the SIOC card were that it be capable of handling serial data transfers to/from the peripheral of either 8-bits plus parity or 16-bits plus parity at selectable bit rates (110, 300, 1200, 9600, 250K, and 500K bits per second). Secondly, all inputs should present no more than one standard TTL load to the peripheral with all outputs being capable of driving one standard TTL load. To reduce the number of control lines between the computer and peripheral, it was decided that all inputs to the computer would be initiated by the peripheral and all outputs from the computer would be initiated by the computer. Last, but not least, the SIOC card was to be low in chip count and power consumption, and run on only 5 VDC.

The requirement of 8-bits plus parity or 16-bits plus parity, along with the transfer rate of 500K bits/second, eliminated all existing LSI devices available

commercially at that time. Because of this unavailability, a discrete design consisting of 33 ICs was implemented using LSTTL technology.

A block diagram of the SIOC card is shown in Figure 29. The 16-bit parallel-to-serial/serial-to-parallel shift register consists of two 8-bit 'LS299s that are cascaded to form a 16-bit device. The bit rate generator consists of four 'LS163As, one-and-a-half 'LS112s, 2/4 of an 'LS00, and 2/4 of an 'LS08. Various points along the cascaded counters are picked off and fed into a 'LS151 multiplexer (8 to 1) and then selected by various combinations of logic levels on BSEL1-BSEL3 as shown in Table 12. The parity generator/checker is made up of an 'LS112 J-K flip-flop; one half checks the input parity, the other half generates the parity bit on all outputs. The box titled "CONTROL LOGIC" consists of 20 or so small scale integration (SSI) and medium scale integration (MSI) devices necessary to generate/receive the control lines to/from the I/O controller and peripheral device.

Table 12 lists the signal pin outs of the serial card. The following describes the role of various signals: Inputs are initiated by the peripheral when it sets $\overline{\text{PBLOK}}$ low. The SIOC in response to $\overline{\text{PBLOK}}$ going low, will set $\overline{\text{PIN}}$ low and wait for the peripheral to pulse $\overline{\text{PSER}}$ low. When the $\overline{\text{PSER}}$ pulse is received by the SIOC, $\overline{\text{CRX}}$ is set low on the rising edge of $\overline{\text{PSER}}$. The peripheral is to put data on the DATA IN line within 50 nanoseconds of the falling edge of $\overline{\text{CRX}}$. Within 1.5/(BIT RATE) seconds maximum, the RTX-CLK will start shifting data into the 'LS299s. The rising edge of RTX-CLK is used to shift data into the 'LS299s. The falling edge of RTX-CLK is to be used by the peripheral to shift data onto the DATA IN line. Depending on the 8/16 line into the SIOC, from the I/O controller, the SIOC will generate either 9 (8/$\overline{16}$="1") or 17 (8/16="0") RTX-CLK pulses. The most-significant-bit (MSB) is transmitted first, followed by the remaining bits in descending order, with the last bit being the parity bit. Within 85 nanoseconds after the falling edge of the last RTX-CLK pulse, the SIOC will set $\overline{\text{INREQ}}$ low. When the CPU card is ready to read the data, the I/O controller will set $\overline{\text{INACK}}$ low, enabling the outputs of the 'LS299s, placing the data on the I/O data bus (IODB0-IODB15). For 8-bit transfers, the 8 MSBs (IODB8-IODB15) will contain all "0s." For 16-bit transfers, the data bus will contain whatever data was transferred. $\overline{\text{INACK}}$ being low will reset the ME bit on the SIOC low if any parity error had occurred on the input transfer. $\overline{\text{INACK}}$ being low is also used by the SIOC to set $\overline{\text{INREQ}}$ back to a "1." When the CPU card has read the data, the I/O controller will set $\overline{\text{INACK}}$ to a high logic level. $\overline{\text{INACK}}$ going high will reset $\overline{\text{CRX}}$ to a high logic level. This procedure will repeat, from $\overline{\text{PSER}}$ being pulsed, until all the words in the block have been transferred. When the block (no limit on number of words) has been transferred, the peripheral will set $\overline{\text{PBLOK}}$ high and the SIOC will respond to the I/O controller by taking $\overline{\text{PIN}}$ high and resetting $\overline{\text{FSTWD}}$ to a low level.

Outputs are initiated by the CPU card setting a bit ($\overline{\text{COUT}}$) on the I/O controller to a low level. The SIOC sees $\overline{\text{COUT}}$ being low and sets $\overline{\text{CBLOK}}$ low. When the CPU card starts writing to the SIOC the I/O controller sets $\overline{\text{WRITEEN}}$ low, and places data on the I/O data bus. A falling edge of $\overline{\text{MCLK}}$, with $\overline{\text{WRITEEN}}$ low,

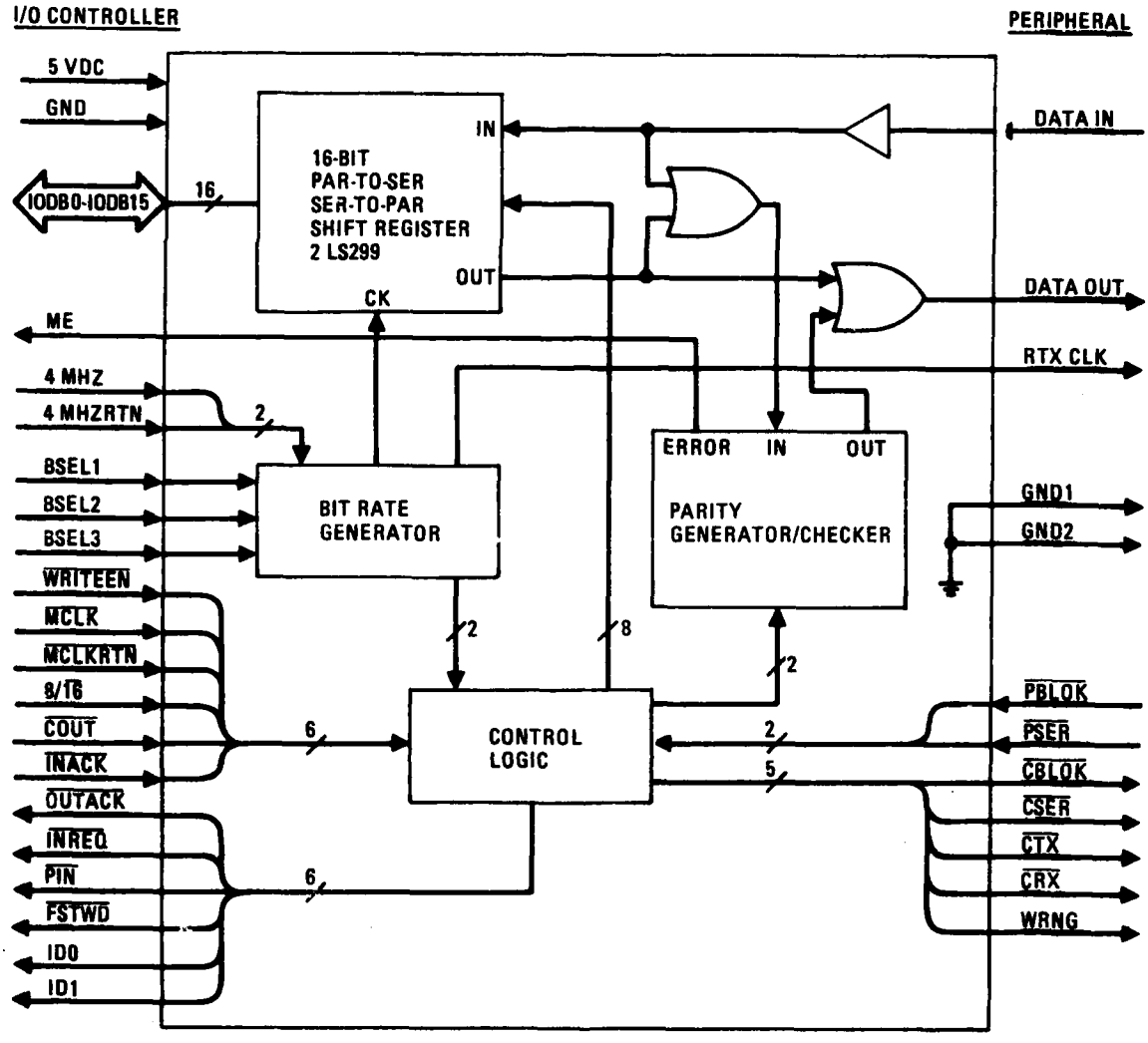


Figure 29. Serial I/O Card Block Diagram

TABLE 12. SERIAL I/O CARD PIN DESCRIPTION

Name	I/O	Description
IODB0-IODB15	I/O	IODB0 (LSB) through IODB15 (MSB) comprise the bidirectional tri-state data bus to the CUP and I/O controller.
ME	OUT	Message error. When active (logic level high), ME indicates to the CPU that the input data word has a parity error. ME is cleared when INACK goes low.
4MHZ	IN	4 MHz clock. It is used to generate the bit rates under program control.

TABLE 12. SERIAL I/O CARD PIN DESCRIPTION (CONTINUED)

Name	I/O	Description
BSEL1-BSEL3	IN	Bit rate select 1 (LSB) through Bit Rate Select 3 (MSB) selects the bit transfer rate or reset function.

BSEL3	BSEL2	BSEL1	Rate/Function
0	0	0	Reset
0	0	1	Not used
0	1	0	500K
0	1	1	250K
1	0	0	9600
1	0	1	1200
1	1	0	300
1	1	1	110

$\overline{\text{WRITEEN}}$	IN	Write enable. When active (logic level low), $\overline{\text{WRITEEN}}$ enables the two LS299s so that a word can be loaded into them. $\overline{\text{WRITEEN}}$ also clears $\overline{\text{OUTACK}}$ when in the active state.
$\overline{\text{MCLK}}$	IN	Memory clock. The falling edge of $\overline{\text{MCLK}}$ is used to clock data from the CPU into the two LS299s for output to the peripheral. $\overline{\text{MCLK}}$ is also used to generate $\overline{\text{CSER}}$ for an output transfer.
$\overline{\text{MCLKRTN}}$	IN	Memory clock return. $\overline{\text{MCLKRTN}}$ is the signal return line from the CPU card. $\overline{\text{MCLK}}$ and $\overline{\text{MCLKRTN}}$ are a twisted pair for better signal shaping.
8/$\overline{16}$	IN	Eight/$\overline{\text{sixteen}}$. Indicates what word length is to be transferred; high = 8 bits, low = 16 bits. If 8/$\overline{16}$ is high, IODB8-IODB15 will have a logic level low placed on them for input transfers.
$\overline{\text{COUT}}$	IN	Computer out. When active (logic level low), $\overline{\text{COUT}}$ indicates that a data transfer to the peripheral via the SIOC is to take place or is presently taking place. $\overline{\text{COUT}}$ going inactive will clear the $\overline{\text{CSER}}$ disable flip-flop, clear $\overline{\text{OUTACK}}$, clear $\overline{\text{CTX}}$, and disable the system clock.
$\overline{\text{INACK}}$	IN	Input acknowledge. When active (logic level low), $\overline{\text{INACK}}$ enables the tri-state buffer, which puts the input data word on the data bus and clears ME. The rising edge of $\overline{\text{INACK}}$ is used to clear $\overline{\text{CRX}}$.

TABLE 12. SERIAL I/O CARD PIN DESCRIPTION (CONTINUED)

Name	I/O	Description
$\overline{\text{OUTACK}}$	OUT	Output acknowledge. When active (logic level low), $\overline{\text{OUTACK}}$ indicates to the CPU that the word to be outputted to the peripheral has been transmitted. $\overline{\text{OUTACK}}$ is cleared by $\overline{\text{WRITEEN}}$ going active or by $\overline{\text{COUT}}$ going inactive.
$\overline{\text{INREQ}}$	OUT	Input request. When active (logic level low), $\overline{\text{INREQ}}$ indicates to the CPU that an input data word is stored in the LS299s. $\overline{\text{INREQ}}$ is cleared by $\overline{\text{INACK}}$ going active.
$\overline{\text{PIN}}$	OUT	Peripheral input. When active (logic level low), $\overline{\text{PIN}}$ indicates that a data transfer into the CPU is taking place or is about to take place.
$\overline{\text{FSTWD}}$	OUT	First word. When active (logic level low), $\overline{\text{FSTWD}}$ indicates that the first word (or only word) of a data transfer to the CPU is to be on the data bus when the CPU enables the data bus with $\overline{\text{INACK}}$ going active. $\overline{\text{FSTWD}}$ is cleared by the rising edge of $\overline{\text{INACK}}$ and set by $\overline{\text{PBLOK}}$ going inactive (logic level high).
ID0-ID1	OUT	ID0 (LSB) = low, ID1 (MSB) = high, indicates to the I/O controller that the I/O card is a programmed handshake card.
DATA IN	IN	This data line carries bit-serial data from the peripheral to the SIOC at the receive-transmit clock frequency.
DATA OUT	OUT	This data line carries bit-serial data from the SIOC to the peripheral at the receive-transmit clock frequency.
RTX-CLK	OUT	Receive/transmit clock. This squarewave clock is used for clocking data in/out of the peripheral at a selected bit rate.
$\overline{\text{PBLOK}}$	IN	Peripheral block. When active (logic level low), $\overline{\text{PBLOK}}$ indicates that a data transfer from the peripheral to the CPU is taking place or is about to take place. $\overline{\text{PBLOK}}$ being active will set $\overline{\text{PIN}}$ active and is used to gate $\overline{\text{INREQ}}$ after an input word is stored in the LS299s. $\overline{\text{PBLOK}}$ being active will reset the first word flip-flop.
$\overline{\text{PSER}}$	IN	Peripheral serial. When pulsed (negative going pulse), $\overline{\text{PSER}}$ signals the SIOC that the peripheral is either capable of receiving or wishes to transmit bit-serial data. $\overline{\text{PSER}}$ is used by the SIOC to set $\overline{\text{CRX}}$ and $\overline{\text{CTX}}$.

TABLE 12. SERIAL I/O CARD PIN DESCRIPTION (CONCLUDED)

Name	I/O	Description
$\overline{\text{CBLOK}}$	OUT	Computer block. When active (logic level low), CBLOK indicates that a data transfer to the peripheral via the SIOC is taking place or is about to take place. CBLOK will go active just before CSER is pulsed and will go inactive just after the rising edge of the last CTX.
$\overline{\text{CSER}}$	OUT	Computer serial. When pulsed (negative going pulse), CSER indicates to the peripheral that the SIOC wishes to send bit-serial data. There will be one CSER pulse each time the SIOC sets CBLOK.
$\overline{\text{CTX}}$	OUT	Computer transmitting. When active (logic level low), CTX indicates to the peripheral that the SIOC is transmitting and will be used in conjunction with RTX-CLK to shift data into the peripheral. CTX is set by the rising edge of PSER and gets cleared after the parity bit is transmitted to the peripheral.
$\overline{\text{CRX}}$	OUT	Computer receiving. When active (logic level low), CRX indicates that the SIOC is receiving and will be used, in conjunction with RTX-CLK, to shift data out of the peripheral. CRX is set by the rising edge of PSER and clear when the CPU reads the byte/word.
WRNG	OUT	Warning. This signal informs the peculiar support equipment (PSE) of the number of bits the SIOC is expecting to transmit or receive. WRNG (logic level high) = 8 bits + 1 parity bit WRNG (logic level low) = 16 bits + 1 parity bit
GND1-GND2	OUT	Ground returns. These pins are brought out to the peripheral connector for ground reference.

is used to latch the data into the two 'LS299s. The first WRITEEN logic level transitions, high to low then low to high, are passed through the SIOC to generate one CSER pulse (negative going pulse) for each output block transfer. Within 50 nanoseconds of the falling edge of MCLK, the SIOC will place the MSB of data on DATA OUT line. When the peripheral is ready to receive data, it will pulse PSER (negative going pulse). The rising edge of PSER is used by the SIOC to set CTX low. Within 1.5/(BIT RATE) seconds, the SIOC will start generating either 9 or 17 (depending on the logic level of 8/16) RTX-CLK pulses. The MSB is transmitted first, followed by the remaining bits in descending order, with the last bit being the parity bit. The SIOC uses the falling edge of RTX-CLK to shift the next data bit onto the DATA OUT line; therefore,

the peripheral is to use the rising edge to latch the data into its shift register. If the CPU card is going to write more words out to the peripheral, the output procedure is repeated when WRITEEN is set low. When WRITEEN goes low, the SIOC will reset CTX to a high level. When the block transfer has been completed, the CPU card will reset the COUT bit on the I/O controller to a high level. This high level is used by the SIOC to set CTX and CBLOK to a high logic level.

As per contract requirements, the SIOC interface to peripheral devices is based on single ended TTL driver and receiver specifications. This gives the advantage of low power consumption on the card; however, it makes operation with cables over two feet long highly unreliable (future versions of the card should incorporate differential line drivers and receivers to relieve this shortcoming and allow the peripheral to be located more than two feet from the computer).

Parallel Input/Output Card

The parallel input-output controller (PIOC) card was designed to act as an interface between the computer and a peripheral device. Programmed handshake was chosen as the type of protocol, rather than direct-memory-access, so that the CPU card could do real-time data manipulation without having to read the data from memory.

The main design goals for the PIOC were that it be capable of handling transfers of either 8-bit bytes or 16-bit words at a rate greater than 250,000 words per second over a bidirectional data bus to a peripheral device in a half-duplex mode. Secondly, all inputs should present no more than one standard TTL load to the peripheral with all outputs being capable of driving one standard TTL load. To reduce the number of control lines between the computer and peripheral, it was decided that all inputs to the computer would be initiated by the peripheral and all outputs from the computer would be initiated by the computer. Last, but not least, the PIOC was to be low in chip count and power consumption, and run on only 5 VDC.

To meet the low chip count and low power consumption requirements, a search was executed for LSI devices that would be capable of meeting the handshake protocol. Because there were no commercially available devices that came close to our I/O controller or peripheral protocols, a discrete design was implemented around two AM 2950 8-bit parallel I/O ports.

A block diagram of the PIOC is shown in Figure 30. The 16-bit input and output data buffers are made up of two AM 2950 8-bit parallel I/O ports. These devices, when connected together, provide a temporary store for one 16-bit word in each direction (two registers of 16 bits each) that are interconnected to provide a bidirectional buffer. The outputs of the input and output data buffers are tri-stated on the AM 2950s, which in effect gives us a buffered-bidirectional 16-bit data bus between the peripheral and I/O controller. The box titled "CONTROL LOGIC" consists of 12 SSI and MSI devices that are required to interface the AM 2950s to the I/O controller and peripheral.

AD-A119 380 GENERAL DYNAMICS SAN DIEGO CA CONVAIR DIV F/8 9/2
DIGITAL INTEGRATING SUBSYSTEM (DIS).(U)
OCT 81 K D ARNOLD, D L BURGESS, J F FASSEL F08635-79-C-0206
UNCLASSIFIED CS64-32228 AFATL-TR-81-83 NL

END
DATE FILMED
10 82
DTIC

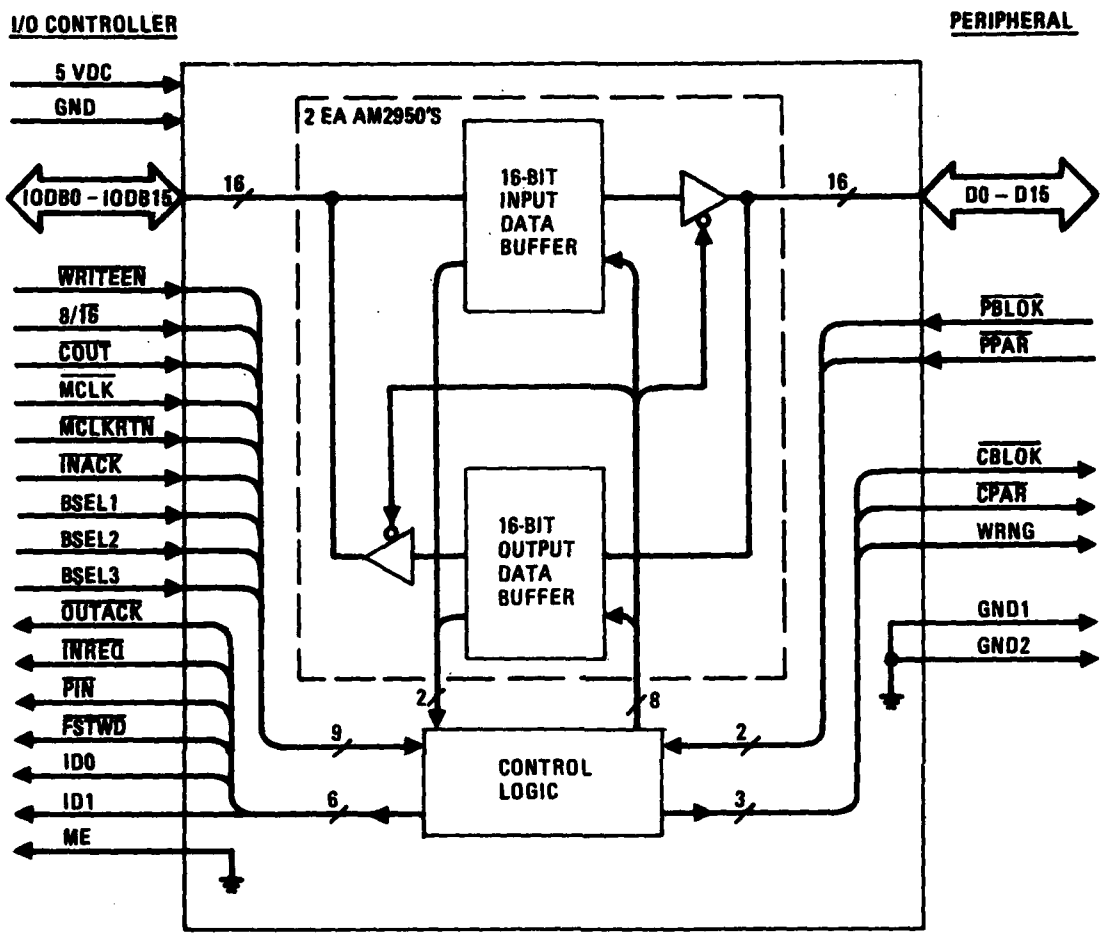


Figure 30. Parallel I/O Card Block Diagram

Table 13 lists the signal pin outs of the parallel card. The following describes the role of the various signals:

TABLE 13. SIGNAL PINOUTS OF PARALLEL CARD

Name	I/O	Description
IODB0-IODB15	I/O	IOCB0 (LSB) through IODB15 (MSB) comprise the bidirectional tri-state data bus to the CPU and I/O controller.
$\overline{\text{WRITEEN}}$	IN	Write enable. When active (logic level low), $\overline{\text{WRITEEN}}$ is used to enable the clock for the output buffer. $\overline{\text{WRITEEN}}$ also clears $\overline{\text{OUTACK}}$ when more than one word is transmitted in any block output.

TABLE 13. SIGNAL PINOUTS OF PARALLEL CARD (CONTINUED)

Name	I/O	Description
8/$\overline{16}$	IN	Eight/$\overline{\text{sixteen}}$. Indicates what word length is to be transferred; high = 8 bits, low = 16 bits. If 8/$\overline{16}$ is high, IODB8-IODB15 will have a logic level low placed on them for input transfers.
$\overline{\text{COUT}}$	IN	Computer out. When active (logic level low), $\overline{\text{COUT}}$ indicates that a data transfer to the peripheral via the PIOC is to take place. $\overline{\text{COUT}}$ is used to generate CBLOK and will clear OUTACK at the end of a block transfer by going to a logic level high.
$\overline{\text{MCLK}}$	IN	Memory clock. The falling edge of $\overline{\text{MCLK}}$ is used to clock data into the output data buffer. $\overline{\text{MCLK}}$ is also used to generate $\overline{\text{CPAR}}$ for output transfers.
$\overline{\text{MCLKRTN}}$	IN	Memory clock return. $\overline{\text{MCLKRTN}}$ is the signal return line from CPU card. $\overline{\text{MCLK}}$ and $\overline{\text{MCLKRTN}}$ are a twisted pair for better signal shaping.
$\overline{\text{INACK}}$	IN	Input acknowledge. When active (logic level low), $\overline{\text{INACK}}$ enables the input data buffer's tri-state bus, which then places the input data on the bus. The rising edge of $\overline{\text{INACK}}$ is used to clear $\overline{\text{INREQ}}$.
BSEL1-BSEL3	IN	Bit rate select lines. When these three lines are all logic level low, a $\overline{\text{RESET}}$ signal is generated to initialize the PIOC.
$\overline{\text{OUTACK}}$	OUT	Output acknowledge. When active (logic level low), OUTACK indicates to the CPU that the peripheral has accepted the data word. OUTACK is cleared by either $\overline{\text{COUT}}$ going inactive (logic level high) or by WRITEEN going active (logic level low).
$\overline{\text{INREQ}}$	OUT	Input request. When active (logic level low), $\overline{\text{INREQ}}$ indicates that a word is stored in the input data buffer. $\overline{\text{INREQ}}$ is cleared by the rising edge of $\overline{\text{INACK}}$.
$\overline{\text{PIN}}$	OUT	Peripheral input. When active (logic level low) $\overline{\text{PIN}}$ indicates that a data transfer into the CPU is taking place or is about to take place.
$\overline{\text{FSTWD}}$	OUT	First word. When active (logic level low), $\overline{\text{FSTWD}}$ indicates that the first word (or only word) of a data transfer to the CPU is to be on the data bus when the CPU enables the data bus. FSTWD is cleared by the

TABLE 13. SIGNAL PINOUTS OF PARALLEL CARD (CONCLUDED)

Name	I/O	Description
		rising edge of $\overline{\text{INACK}}$ and set low when $\overline{\text{PBLOK}}$ goes inactive (logic level high).
ID0-ID1	OUT	ID0 (LSB) = low, ID1 (MSB) = high, indicates to the I/O controller that the I/O card is a programmed handshake card.
ME	OUT	Message error. Message error is tied low so that the I/O controller of CPU doesn't sense a bit error has occurred on the PIOC.
D0-D15	I/O	D0 (LSB) through D15 (MSB) comprise the bidirectional tri-state data bus to the peripheral.
$\overline{\text{PBLOK}}$	IN	Peripheral block. When active (logic level low), $\overline{\text{PBLOK}}$ indicates that a data transfer to the CPU via the PIOC is to take place. $\overline{\text{PBLOK}}$ is used to generate $\overline{\text{PIN}}$ and to reset $\overline{\text{FSTWD}}$.
$\overline{\text{PPAR}}$	IN	Peripheral parallel acknowledge. When pulsed (negative going pulse), $\overline{\text{PPAR}}$ indicates that either valid information is available to the PIOC from the peripheral or that the information from the PIOC has been accepted.
$\overline{\text{CBLOK}}$	OUT	Computer block. When active (logic level low), $\overline{\text{CBLOK}}$ indicates either a data transfer to the peripheral is taking place or is to take place. $\overline{\text{CBLOK}}$ is generated by $\overline{\text{COUT}}$ going active (logic level low).
$\overline{\text{CPAR}}$	OUT	Computer parallel acknowledge. When pulsed (negative going pulse), $\overline{\text{CPAR}}$ indicates that valid information is available to the peripheral or that the information from the peripheral has been accepted.
WRNG	OUT	Warning. This signal informs the peculiar support equipment (PSE) of the number of bits the PIOC is expecting to transmit or receive. WRNG (logic level high = 8 bits WRNG (logic level low) = 16 bits
GND1-GND2	OUT	Ground returns. These pins are brought out to the peripheral connector for ground reference.

Inputs are initiated by the peripheral when it sets $\overline{\text{PBLOK}}$ low. The PIOC at this time sets $\overline{\text{PIN}}$ low, which informs the input/output controller (I/OC) that an input transfer is about to occur. The peripheral places data on the bus (D0-D15)

and pulses PPAR (negative going pulse). The falling edge of PPAR latches data into the input data buffer, and the data latched flags from the AM 2950s are NANDed to generate INREQ. When the CPU card is ready to read the data, the I/OC will set INACK low, enabling the input data buffer on the PIOC, placing data on the data bus (IODB0-IODB15). CPAR at this time will also go low. When the CPU card has read the data, INACK will go high. INACK going high will disable the tri-stated outputs of the input data buffer, reset INREQ, clear FSTWD, and set CPAR high again. This procedure will repeat, from PPAR being pulsed, until all the words in the block have been transferred. When the block (no limit on number of words) has been transferred, the peripheral will set PBLOK high and the PIOC will respond to the I/OC by taking PIN high and resetting FSTWD to a low level.

Outputs are initiated by the CPU card setting a bit (COUT) on the I/OC to a low level. The PIOC sees COUT going low and sets CBLOK low. Then the CPU card starts writing to the PIOC, the I/OC sets WRITEEN low, places data on the data bus (IODB0-IODB15), and sets MCLK low. The falling edge of MCLK latches the data into the output data buffer. At this time, the output data buffer places the latched data onto the peripheral data bus. The PIOC allows one inverted MCLK pulse to go through the PIOC to the peripheral in the form of a CPAR pulse (negative going pulse). When the peripheral has read the data, it will pulse PPAR setting OUTACK low. This procedure will repeat, from WRITEEN going low (OUTACK is reset high at this time), until all words of the block have been transferred. The CPU will set COUT high and the PIOC will then set CBLOK and OUTACK high in response.

As per contract requirements, the PIOC interface to peripheral devices is based on single ended TTL driver and receiver specification. This gives the advantage of low power consumption on the card; however, it makes operation with cables over 2 feet long highly unreliable (future versions of this card should incorporate differential line drivers and receivers to relieve this shortcoming and allow the peripheral to be located more than 2 feet from the I/O card).

Direct Memory Access Input/Output Card

The direct memory access (DMA) I/O card was designed for block transfers of data (up to 256 words in any one block) between a peripheral device and the computer memory without CPU intervention. Of the two common methods of transferring input/output data, program controlled and DMA, program controlled I/O is particularly useful for low speed interface requirements, as well as for dedicated peripheral applications. However, for microcomputer systems where both processing and I/O operations are required to operate at medium to high speeds, the DMA method is preferable.

The main design goals for the DMA I/O card were that it be capable of handling transfers of either 8-bit bytes or 16-bit words at a rate greater than 500,000 words per second over a bidirectional data bus to/from a peripheral device in a half-duplex mode. These data transfers were to be transparent to the CPU by

being under control of a DMA controller (DMAC), and work in a cycle-stealing mode rather than halting the processor. The DMA I/O card is to pass a 6-bit block ID, from the peripheral to the DMAC, which points the DMAC to a location in its small memory bank where the start address is located. The DMA I/O card is to respond within 2 microseconds to a request with an acknowledge. All inputs on the DMA I/O card should present no more than one standard TTL load to the peripheral with all outputs being capable of driving one standard TTL load. Inputs can be initiated by either the peripheral or the CPU, while outputs are initiated by only the CPU. Finally, the DMA I/O card was to be low in chip count, power consumption, and run on a single 5 VDC supply.

A block diagram of the DMA I/O card is shown in Figure 31. The 16-bit input and output data buffers are made up of two AM 2950 8-bit parallel I/O ports. These devices, when connected together, provide a temporary store for one 16-bit word in each direction (two registers of 16-bits each) that are interconnected to provide a bidirectional buffer. The outputs of the input and output data buffers are tri-stated onboard the AM 2950s, which in effect gives us a buffered-bidirectional 16-bit data bus between the peripheral and I/O controller. The box titled "CONTROL LOGIC" consists of 13 SSI and MSI devices that are required to interface the AM 2950s to the I/O controller and peripheral.

Table 14 lists the signal pin outs of the DMA card. The following describes the role of the various signals.

Inputs can be initiated by either the peripheral or the CPU card. The peripheral initiates DMA inputs by placing a valid 6-bit block ID on the ID lines, sets PINREQ low, and places data on the peripheral data bus (D0-D15). The DMA I/O card responds to PINREQ going low by setting PIN low. The peripheral now pulses PACK. The falling edge of PACK latches the data into the input data buffer, and the data latched flags from the AM 2950s are NANDed to generate DINREQ. The I/OC will wait for a DMACK from the CPU card before proceeding. When the I/OC is granted the memory bus, it will read the block ID, set INACK low, enabling the input data buffer on the DMA I/O card, placing data on the data bus (IODB0-IODB-15). INACK going low is used by the DMA I/O card to set CACK low. When the data has been written into memory by the I/OC, INACK will go high. INACK going high will disable the tri-stated outputs of the input data buffer, reset DINREQ, clear FSTWD, and set CACK high again. This procedure will be repeated, from PACK being pulsed, until all the words in the block have been transferred. When the block (there is a systems limit of 256 words per block; however, the hardware has no limit) has been transferred, the peripheral will set PINREQ high and the DMA I/O card will respond by setting PIN high and resetting FSTWD to a low level.

When the CPU is required to initiate a DMA input, it will inform the DMAC, located on the I/OC, by setting a flag for whatever I/O slot it wants the data from. The I/OC will set CIN low, and the DMA I/O card will respond by setting CINREQ low. When the peripheral senses CINREQ going low, it will respond by initiating a DMA input as described in the last paragraph. When the I/OC sees

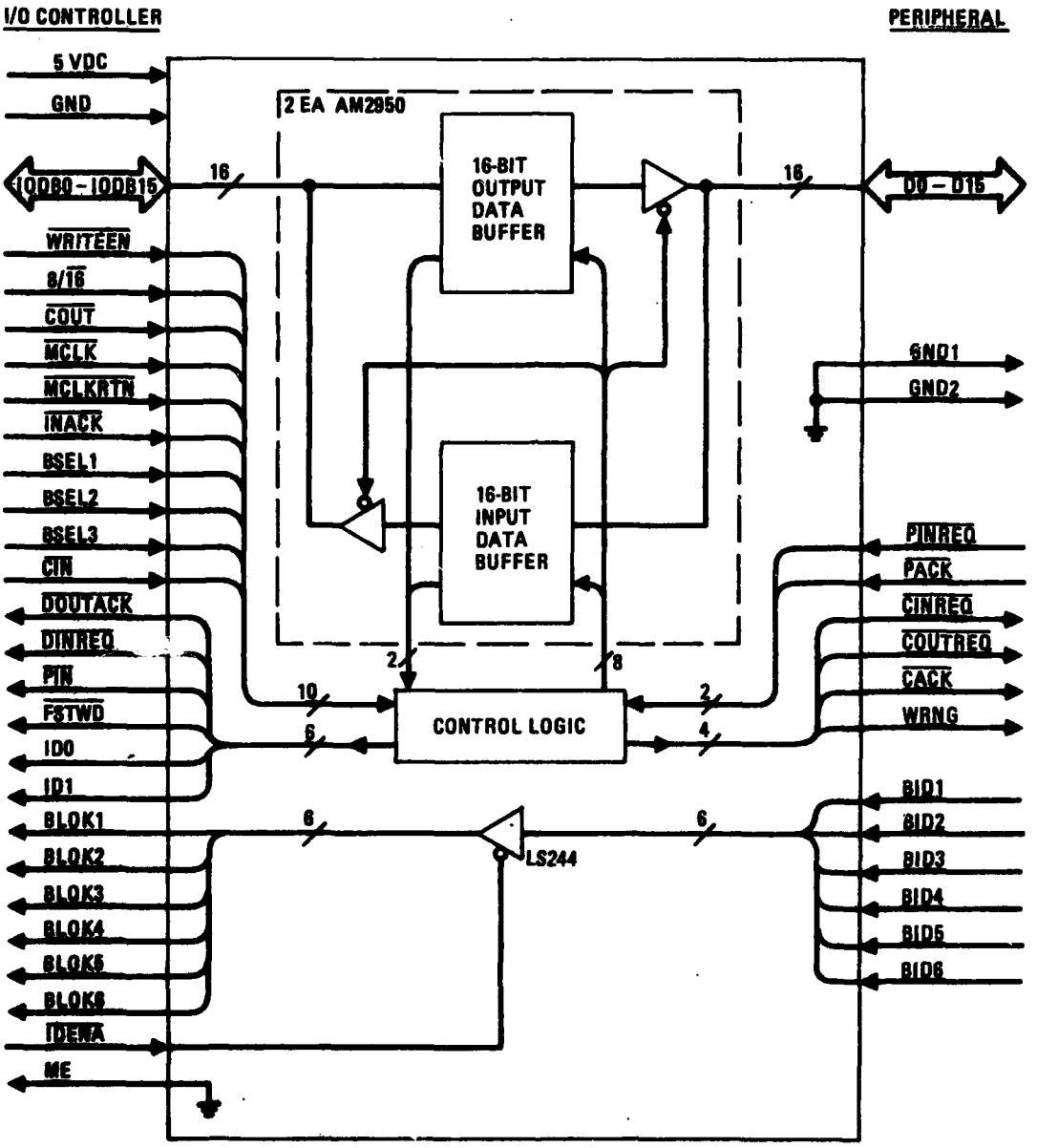


Figure 31. DMA I/O Card Block Diagram

the first $\overline{\text{DINREQ}}$ going low, it will reset $\overline{\text{CIN}}$ to a high level, which in turn will get $\overline{\text{CINREQ}}$ back to a high logic level.

Outputs are initiated only by the CPU card. The CPU will load the DMAC with the start address, word count, and then a control word that tells it what slot to do the DMA through. The DMAC then sets $\overline{\text{COUT}}$ low. The DMA I/O card sees $\overline{\text{COUT}}$ going low and sets $\overline{\text{COUTREQ}}$ low. When the DMAC starts writing to the I/O card, the DMAC sets $\overline{\text{WRITEEN}}$ low and places data on the data bus

TABLE 14. SIGNAL PINOUTS OF DMA CARD

Name	I/O	Description
IODB0-IODB15	I/O	IODB0 (LSB) through IODB15 (MSB) comprise the bidirectional tri-state data bus to the DMAC and I/O controller.
WRITEEN	IN	Write enable. When active (logic level low), WRITEEN is used to enable the clock for the output data buffer. WRITEEN also clears DOUTACK when more than one word is transmitted in any block output.
8/16	IN	Eight/sixteen. Indicates what word length is to be transferred; high = 8 bits, low = 16 bits. If 8/16 is high, IODB8-IODB15 will have a logic level low placed on them for input transfers.
COUT	IN	Computer out. When active (logic level low), COUT indicates that a data transfer to the peripheral via the DMA card is to take place. COUT is used to generate COUTREQ and will clear DOUTACK at the end of a block transfer by going to a logic level high.
MCLK	IN	Memory clock. The falling edge of MCLK is used to clock data into the output data buffer. MCLK is also used to generate CACK for output answers.
MCLKRTN	IN	Memory clock return. MCLKRTN is the signal return line from CPU card. MCLK and MCLKRTN are a twisted pair for better signal shaping.
INACK	IN	Input acknowledge. When active (logic level low), INACK enables the input data buffer's tri-state bus, which then places the input data on the bus. The rising edge of INACK is used to clear DINREQ.
BSEL1-BSEL3	IN	Bit rate select lines. When these three lines are all logic level low, a reset signal is generated to initialize the DMA I/O card.
CIN	IN	Computer in. When active (logic level low), CIN is used to generate CINREQ.
DOUTACK	OUT	DMA output acknowledge. When active (logic level low), DOUTACK indicates to the DMAC that the peripheral has accepted the data word. DOUTACK is cleared by either COUT going inactive (logic level high) or by WRITEEN going active (logic level low).
DINREQ	OUT	DMA input request. When active (logic level low), DINREQ indicates that a word is stored in the input data buffer. DINREQ is cleared by the rising edge of INACK.

TABLE 14. SIGNAL PINOUTS OF DMA CARD (CONTINUED)

Name	I/O	Description
$\overline{\text{PIN}}$	OUT	Peripheral input. When active (logic level low), $\overline{\text{PIN}}$ indicates that a data transfer into the memory is taking place or is about to take place.
$\overline{\text{FSTWD}}$	OUT	First word. When active (logic level low), $\overline{\text{FSTWD}}$ indicates that the first word (or only word) of a data transfer to the memory is to be on the data bus when the DMAC enables the data bus. $\overline{\text{FSTWD}}$ is cleared by the rising edge of INACK and set low when $\overline{\text{PINREQ}}$ goes inactive (logic level high).
ID0-ID1	OUT	ID0 (LSB) = high, ID1 (MSB) = low, indicates to the I/O controller that the I/O card is a DMA I/O card.
BLOK1-BLOK6	OUT	Block ID1 through block ID6 are used to point the DMAC to a start address and word count for a DMA input transfer.
$\overline{\text{IDENA}}$	IN	ID enable. When active (logic level low) $\overline{\text{IDENA}}$ places the block ID on the tri-state block ID bus
ME	OUT	Message error. Message error is tied low so that the I/O controller or CPU doesn't sense a bit error has occurred on the DMA I/O card.
D0-D15	I/O	D0 (LSB) through D15 (MSB) comprise the bidirectional tri-state data bus to the peripheral.
$\overline{\text{PINREQ}}$	IN	Peripheral input request. When active (logic level low), $\overline{\text{PINREQ}}$ indicates that a data transfer to the memory via the DMA I/O card is to take place. $\overline{\text{PINREQ}}$ is used to generate $\overline{\text{PIN}}$ and to reset $\overline{\text{FSTWD}}$.
$\overline{\text{PACK}}$	IN	Peripheral acknowledge. When pulsed (negative going pulse), $\overline{\text{PACK}}$ indicates that either valid information is available to the DMA card from the peripheral or that the information from the DMA card has been accepted.
$\overline{\text{COUTREQ}}$	OUT	Computer output request. When active (logic level low), $\overline{\text{COUTREQ}}$ indicates either a data transfer to the peripheral is taking place or is about to take place. $\overline{\text{COUTREQ}}$ is generated by $\overline{\text{COUT}}$ going active (logic level low).
$\overline{\text{CACK}}$	OUT	Computer acknowledge. When pulsed (negative going pulse), $\overline{\text{CACK}}$ indicates that valid information is available to the peripheral or that the information from the peripheral has been accepted.

TABLE 14. SIGNAL PINOUTS OF DMA CARDS (CONCLUDED)

Name	I/O	Description
BID1-BID6	IN	BID1 (LSB) through BID6 (MSB) are the block ID bits from the peripheral.
$\overline{\text{CINREQ}}$	OUT	Computer input request. When active (logic level low), $\overline{\text{CINREQ}}$ requests a DMA input from the peripheral. $\overline{\text{CINREQ}}$ will go inactive when the DMAC recognizes an input request from the peripheral.
WRNG	OUT	Warning. This signal informs the peculiar support equipment (PSE) of the number of bits the DMA I/O card is expecting to transmit or receive: WRNG (logic level high) = 8 bits WRNG (logic level low) = 16 bits
GND1-GND2	OUT	Ground returns. These pins are brought out to the peripheral connector for ground reference.

IODB0-IODB15). The falling edge of $\overline{\text{MCLK}}$ at the end of the DMA cycle latches the data into the output data buffer. At this time, the output data buffer places the latched data onto the peripheral data bus. The DMA I/O card allows one inverted MCLK pulse to go through itself to the peripheral in the form of a $\overline{\text{CACK}}$ pulse (negative going pulse). When the peripheral has read the data, it will pulse $\overline{\text{PACK}}$, which will set $\overline{\text{DOUTACK}}$ low. This procedure will repeat, from $\overline{\text{WRITEEN}}$ going low ($\overline{\text{DOUTACK}}$ is reset at this time), until all words of the block have been transferred. The DMAC will then set $\overline{\text{COUT}}$ high and the DMA I/O card will set $\overline{\text{COUTREQ}}$ and $\overline{\text{DOUTACK}}$ high in response.

MIL-STD-1553B I/O Card

The MIL-STD-1553B I/O card was designed to provide an umbilical link between an aircraft digital communications bus and a low-cost tactical missile. This link is to be MIL-STD-1553B and MIL-STD-1760 compatible to standardize interface hardware to the launch vehicle.

The MIL-STD-1553B data bus is a serial, time division multiplexed, digital transmission system. Remote terminal units (RTUs) and a control terminal unit (CTU) are connected to each other via a shielded twisted pair cable. The CTU (in the launch vehicle) directs signal traffic on the bus. RTUs respond to commands from the CTU. The missile end of the umbilical link is the 1553B I/O card acting as a RTU.

The I/O card was designed to the MIL-STD-1553B specifications. Some of the major design requirements are:

a. Information is transferred via:

   1. Controller terminal to remote terminal
   2. Remote terminal to controller terminal
   3. Remote terminal to remote terminals
   4. Broadcast

b. A maximum of 32 terminals can be on the bus at any one time, each being capable of transmitting 64 types of messages.

c. The maximum transmission is 32 data words in serial digital pulse code form, Manchester bi-phase at a one megabit per second rate.

d. A dual redundant link is possible by using two of these I/O cards.

e. Transformer coupling is used (direct stub).

f. The bus voltage at the input of the stub is between 18V-27V peak-peak (the characteristic impedance of the line is 77 ohms).

Because the MIL-STD 1553B protocol is so similar to the BIU protocol, the final designs of each card were very similar. Therefore the trade studies that were pursued for the BIU investigation were also applicable for this I/O card.

The final circuit design is shown in block diagram form in Figure 32. The MIL-STD-1553B card has been designed to interface between the actual 1553B bus and the I/O controller. The 12H6 PAL is used for combinational logic as per the logic equations of Figure 33. The HEX table for this PAL is shown in Figure 34.

The registered PROM (AM 27525 is the micro-machine providing eight bits of output code to enable various portions of card circuitry. Figure 35 shows the actual HEX output code for the machine states.

The operation of the card is best described from the flow diagram of the protocol shown in Figure 36. On power up, the card is reset and waiting for a command from the bus. The card's micro-machine will respond with a unique output pattern depending on the type of command received. If a receive command is recognized, the micro-code will respond so that the following events occur:

a. The command is transferred to the controller.

b. $\overline{\text{PIN}}$, $\overline{\text{FSTWD}}$, and $\overline{\text{INREQ}}$ will go low.

c. The block ID will be loaded into the tri-state buffer.

If the next received word is a data word (normal CTU to RTU transfer) the word(s) are transferred one by one to the controller. After the last word, the card will respond with a status word back on the bus (clearing this register

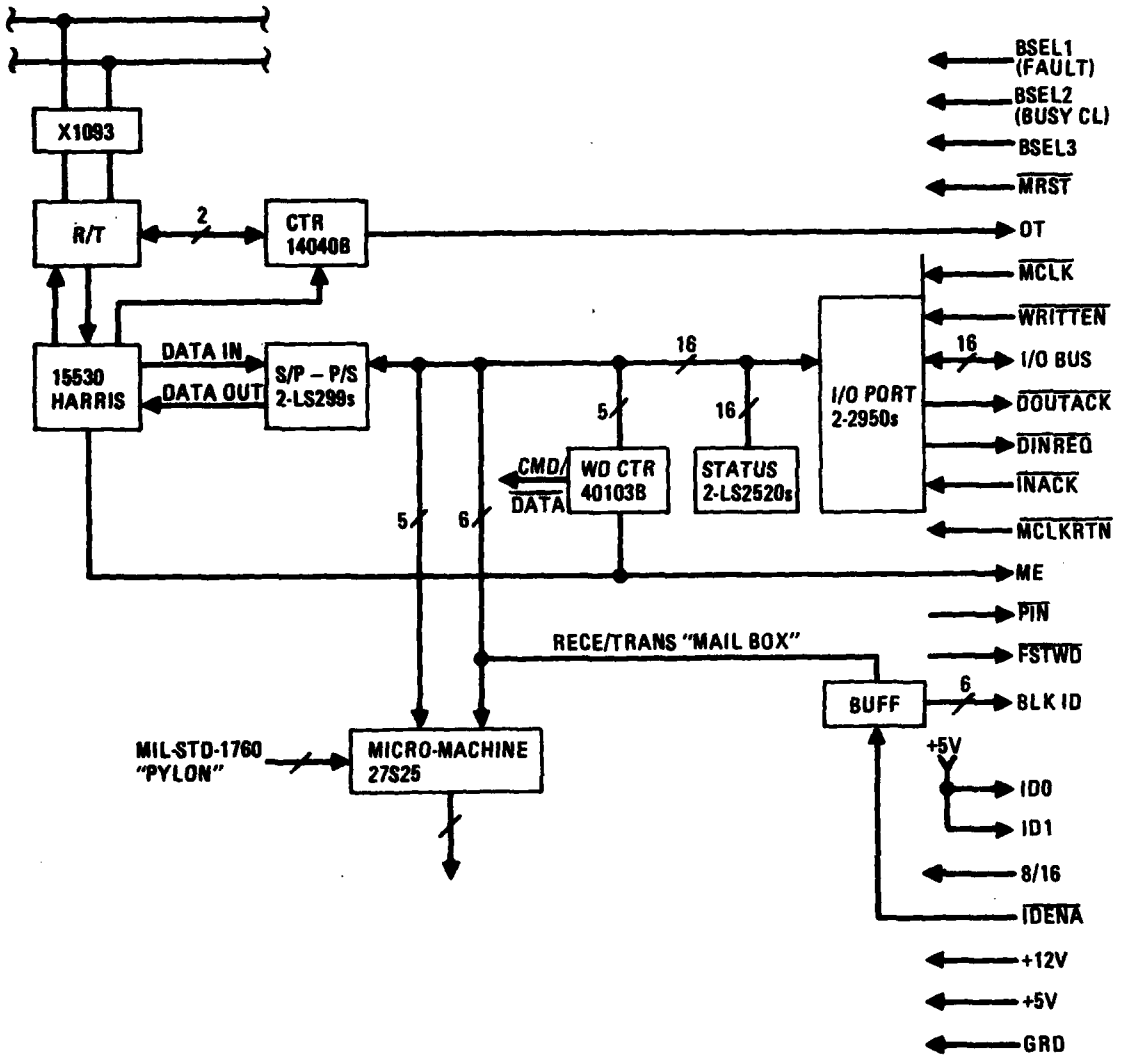


Figure 32. Single Channel 1553B I/O Card Block Diagram

```
PAL12H6
PAT0001

ENCLK SD TD DECLK /MRST 12MHZ 299CLK Z41Q5 Z44Q9 GND /OT DE
299CLK S1 Z94CLK Z56CLK /SD EE SDT-3 VCC

DE = SDIN*/OT + EE*/OT + SDT-3*/OT
299CLK = SDIN*ENCLK + TD*DECLK
S1 = TD + SDIN* /OT + EE* /OT + SDT-3*10T
Z94CLK = /ENCLK*/MRST + MRST*12MHZ
Z56CLK = Z44/Q8*Q7 + ZD
/SD = /SDIN
THIS IS THE END
..
```

Figure 33. Logic Equations for 12H6PAL

BEGINNING ADDRESS	ADDRESS COLUMN 1/ 0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
	OUTPUT WORD 1/ 2/ $0E86															
0000	09	07	0F	0B	0F	0F	0D	0F	0F	0F	0F	0F	0F	0F	0F	0F
0010	0F	0F	0F	0F	0F	0F	0F	0F	0F	0F	0F	0F	0F	0F	0F	0F
0020	07	07	07	07	03	07	05	07	03	07	0F	0F	07	07	0F	0F
0030	07	07	0F	0F	07	07	0F	0F	07	07	05	07	07	07	07	07
0040	03	03	03	03	03	03	01	03	03	03	03	03	03	03	03	03
0050	03	03	03	03	03	03	03	03	03	03	03	03	03	03	01	03
0060	01	01	01	01	01	01	01	01	01	01	03	03	01	01	03	03
0070	01	01	03	03	01	01	03	03	01	01	01	01	01	01	01	01
0080	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01
0090	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01
00A0	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01
00B0	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01
00C0	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01
00D0	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01
00E0	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01
00F0	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01
0100	0F	0F	0E	0F	0B	0F	0F	0F	0F	0F	0F	0F	0F	0F	0F	0F
0110	0F	0F	0F	0F	0D	0F	0F	0F	0D	0F	0F	0F	0F	0F	0F	0F
0120	0B	0F	0F	0F	0F	0F	0B	0F	0F	0F	0F	0F	0F	0E	0F	0F
0130	0E	0F	0F	0F	0F	0F	0F	0F	0F	0F	0F	0F	0D	0F	0F	0F
0140	0C	0C	0C	0C	0C	0C	08	0C	0C	0C	0C	0C	0C	0C	0C	0C
0150	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	08	0C	0C	0C	0C
0160	0C	0C	0C	0C	0C	0C	08	0C	0C	0C	0C	0C	0C	0C	0C	0C
0170	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	08	0C
0180	08	08	08	08	08	08	08	08	08	08	08	08	08	08	08	08
0190	08	08	08	08	08	08	08	08	08	08	08	08	08	08	08	08
01A0	08	08	08	08	08	08	08	08	08	08	08	08	08	08	08	08
01B0	08	08	08	08	08	08	08	08	08	08	08	08	08	08	08	08
01C0	08	08	08	08	08	08	08	08	08	08	08	08	08	08	08	08
01D0	08	08	08	08	08	08	08	08	08	08	08	08	08	08	08	08
01E0	08	08	08	08	08	08	08	08	08	08	08	08	08	08	08	08
01F0	08	08	08	08	08	08	08	08	08	08	08	08	08	08	08	08

NOTES: 1/ INPUT ADDRESSES AND OUTPUT WORDS ARE IN HEXADECIMAL FORMAT.

2/ ADDRESS OF A PARTICULAR OUTPUT WORD IS OBTAINED BY SUBSTITUTING THE LAST DIGIT OF THE BEGINNING ADDRESS WITH THE COLUMN. EXAMPLE: FOR OUTPUT WORD 31, THE ADDRESS IS 001F.

Figure 34. Hex Table for 12H6PAL

	ADDRESS COLUMN 1/																OUTPUT WORDS (REFERENCE)
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
	OUTPUT WORD 1/ 2/																
000	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	000-015
010	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	016-031
020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	032-047
030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	048-063
040	11	11	11	00	11	11	11	00	00	00	11	00	11	00	11	00	064-079
050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	080-095
060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	096-111
070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	112-127
080	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	128-143
090	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	144-159
0A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	160-175
0B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	176-191
0C0	8F	8F	8F	00	8F	8F	8F	00	00	00	8F	00	8F	00	8F	00	192-207
0D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	208-223
0E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	224-239
0F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	240-255
100	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	256-271
110	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	272-287
120	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	288-303
130	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	304-319
140	00	00	0A	00	00	00	00	00	00	00	00	00	00	00	00	00	320-335
150	00	24	00	00	24	6C	00	00	00	00	24	00	6C	00	6C	00	336-351
160	55	55	55	00	55	55	55	00	00	00	55	00	55	00	55	00	352-367
170	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	368-383
180	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	384-399
190	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	400-415
1A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	416-431
1B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	432-447
1C0	00	02	0A	00	00	06	0E	00	00	00	00	00	00	00	00	00	448-463
1D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	464-479
1E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	480-495
1F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	496-511

NOTES: 1/ INPUT ADDRESSES AND OUTPUT WORDS ARE IN HEXADECIMAL FORMAT.
2/ THE ADDRESS OF A PARTICULAR OUTPUT WORD IS OBTAINED BY SUBSTITUTING THE LAST DIGIT OF THE BEGINNING ADDRESS WITH THE COLUMN NUMBER.

Figure 35. Hex Output for Micromachine

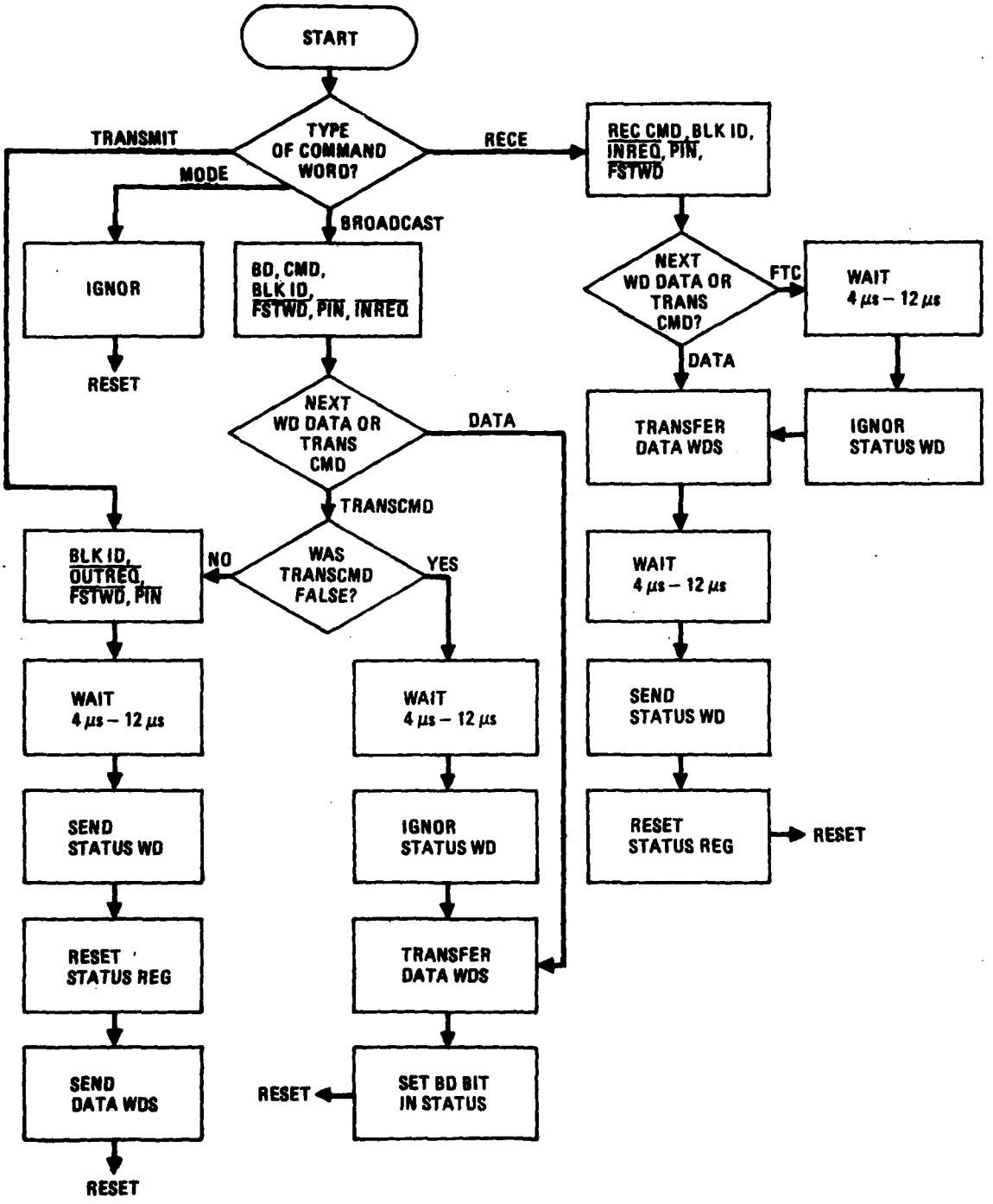


Figure 36. Protocol Diagram

right after transmission). The receive portion is completed when the micro-machine is reset.

If after a valid receive command, the card receives a transmit command to another RTU, this 1553B card will wait, ignore the status word sent by the "other" RTU, and then transfer the data words as before. This extra loop allows this I/O card to receive on RTU to RTU transfers.

If the card is reset to begin with when a broadcast command is recognized, the micro-code will respond so that the following events occur:

a. The command is transferred to the controller;

b. $\overline{\text{PIN}}$, $\overline{\text{FSTWD}}$, and $\overline{\text{INREQ}}$ will go low;

c. The block ID will be loaded into the tri-state buffer.

If the next received word is a data word (as in a normal receive command but this time all RTUs are receiving the data) the word(s) are transferred as before except no status word is sent back; only the broadcast bit is set in the status register.

If, however, a transmit command immediately followed the broadcast command, a decision is made by the machine. If the transmit command is to some other RTU, the micro-machine will cause the card to wait, ignore the status word sent, and then transfer the data words as before. (This allows this card to respond to broadcast commands where another RTU is transmitting.) If the transmit command is for this I/O card, the micro-machine will jump to the transmit routine. (This allows the card to transmit to all other RTUs.)

In the event of a transmit command (either first from reset or during the broadcast chain as previously discussed), the micro-machine will respond so that the following events occur:

a. The block ID is loaded into the tri-state buffer.

b. $\overline{\text{PIN}}$, $\overline{\text{FSTWD}}$, and $\overline{\text{OUTREQ}}$ will go low.

c. The status word is sent out to the bus and reset after transmission.

d. The data words are taken from the controller, one-by-one, and sent out to the 1553 bus.

Once the last data has been sent, the micro-machine resets the card. This type of transfer is of the RTU to CTU type.

As shown in the protocol diagram, this card will not respond to any type of mode code.

Table 15 describes the pin outs of the MIL-STD-1553B card:

TABLE 15. 1553 CARD PIN DESCRIPTION

8/$\overline{16}$:	Used for service request.
$\overline{\text{INACK}}$:	Active low from I/O controller acknowledging input data.
I/O BUS:	16-bit bidirectional parallel bus.
$\overline{\text{DINREQ}}$:	Active low from 1553 card requesting service from I/O controller.
$\overline{\text{WRITEEN}}$:	Active low from I/O controller enabling loading off 1553 output data on MCLK edge.
BLOCK ID:	6-bit encoded lines from 1553 I/O card to I/O controller equivalent to starting address.
BSEL1-3:	If lines equal to zero, reset function (common to all I/O cards) (BSEL2 also acts as busy clear, BSEL1 also acts as Missile Fault).
$\overline{\text{MRST}}$:	Master reset.
$\overline{\text{FSTWD}}$:	Active low from 1553 card signaling controller that first word of transfer is in progress.
$\overline{\text{DOUTACK}}$:	Active low from 1553 card acknowledging previous transmitted word from I/O controller and indicating ready for next word.
OT:	Overtime flag greater than 800 microseconds disables transmitter; resets on master reset.
ME:	Message error flag; active high for error detected on any received word or number of words.
$\overline{\text{MCLK}}$:	Processor clock used in I/O exchange.
$\overline{\text{IDENA}}$:	Active low for controller enabling the block ID on the tri-state controller ID bus.
$\overline{\text{PIN}}$:	Active low during entire transfer of received bus data to controller.

RESULTS

The results of each card design will be discussed in this subsection.

Serial I/O

The serial I/O card was conveniently packaged on 24 square inches as shown in the top level assembly drawing of Figure 37. The circuit design has been

tested and verified for proper operation; however, single ended lines on the peripheral side of the card are susceptible to noise problems.

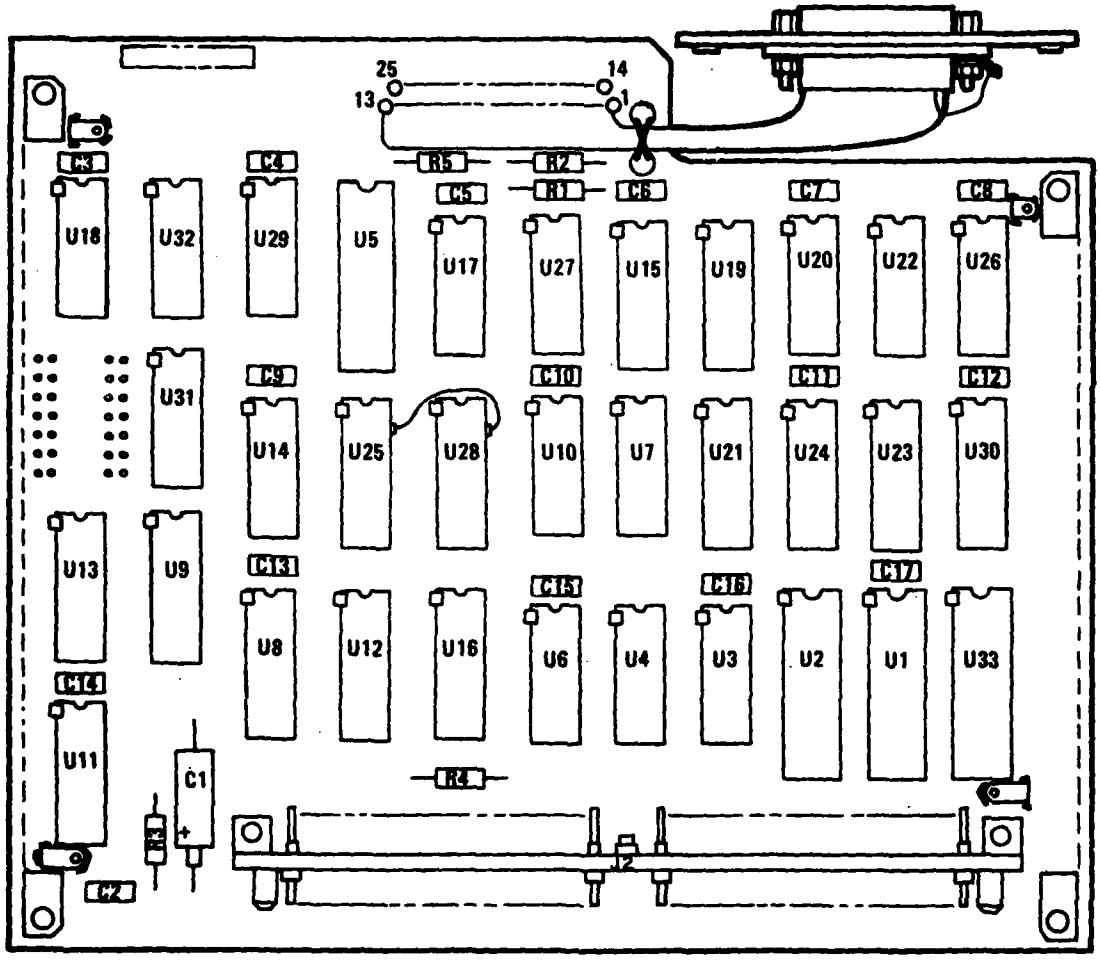


Figure 37. Top Level Assembly Drawing of SIOC

Parallel I/O

The parallel I/O was successfully designed, fabricated, and tested under this contract. The top level assembly drawing is shown in Figure 38. The final card contained all DIPs. The noise problems evident on the serial card were also apparent on this card due to the single ended output lines.

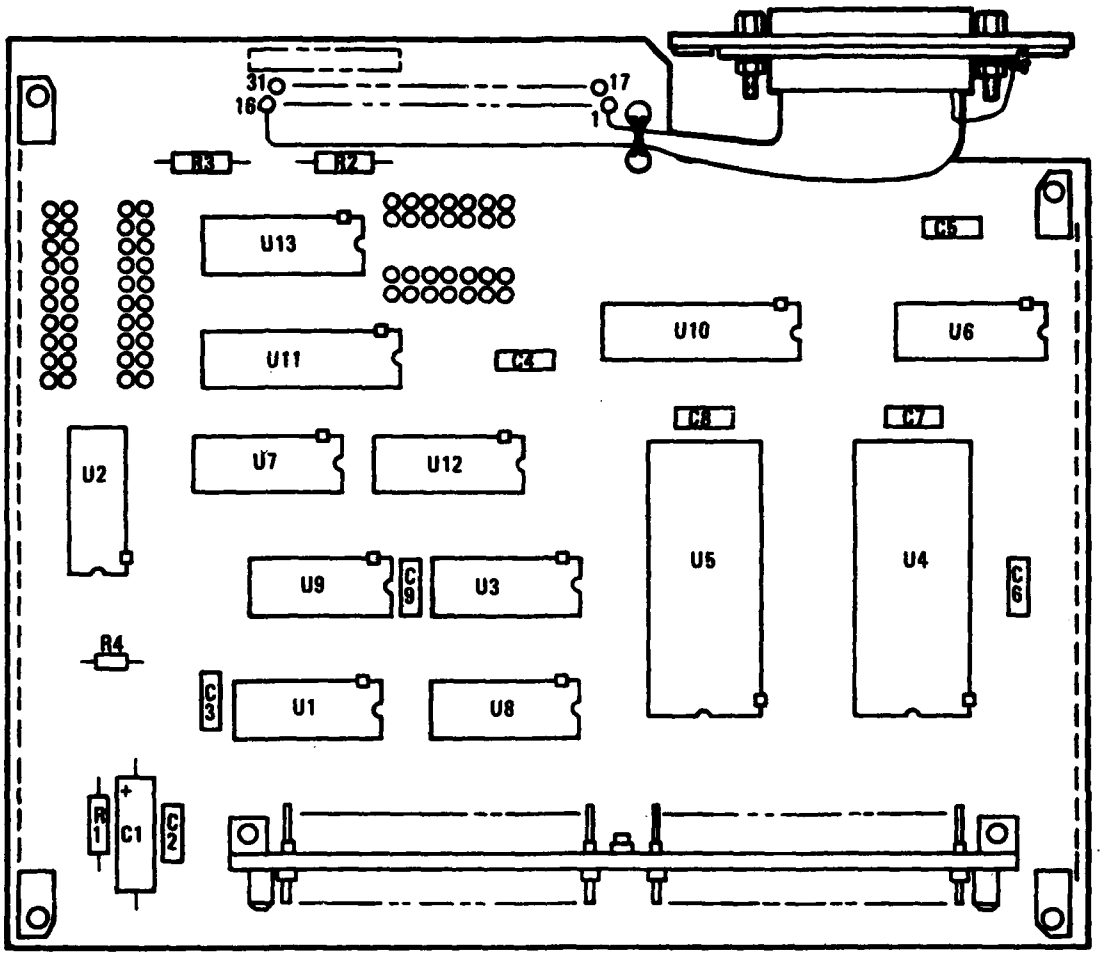


Figure 38. Top Level Assembly Drawing of PIOC

DMA I/O

The DMA design and final card is very similar to the parallel I/O card (the final assembly drawing is shown in Figure 39). Similar noise problems were encountered with this card.

This card provides a different internal interface from the SIOC or PIOC in that those cards are programmed handshake. The DMA card transfers data directly to/from memory interrupting the CPU only once at the end of the block of data transferred. This DMA feature was successfully shown to significantly reduce processing overhead in servicing interrupts.

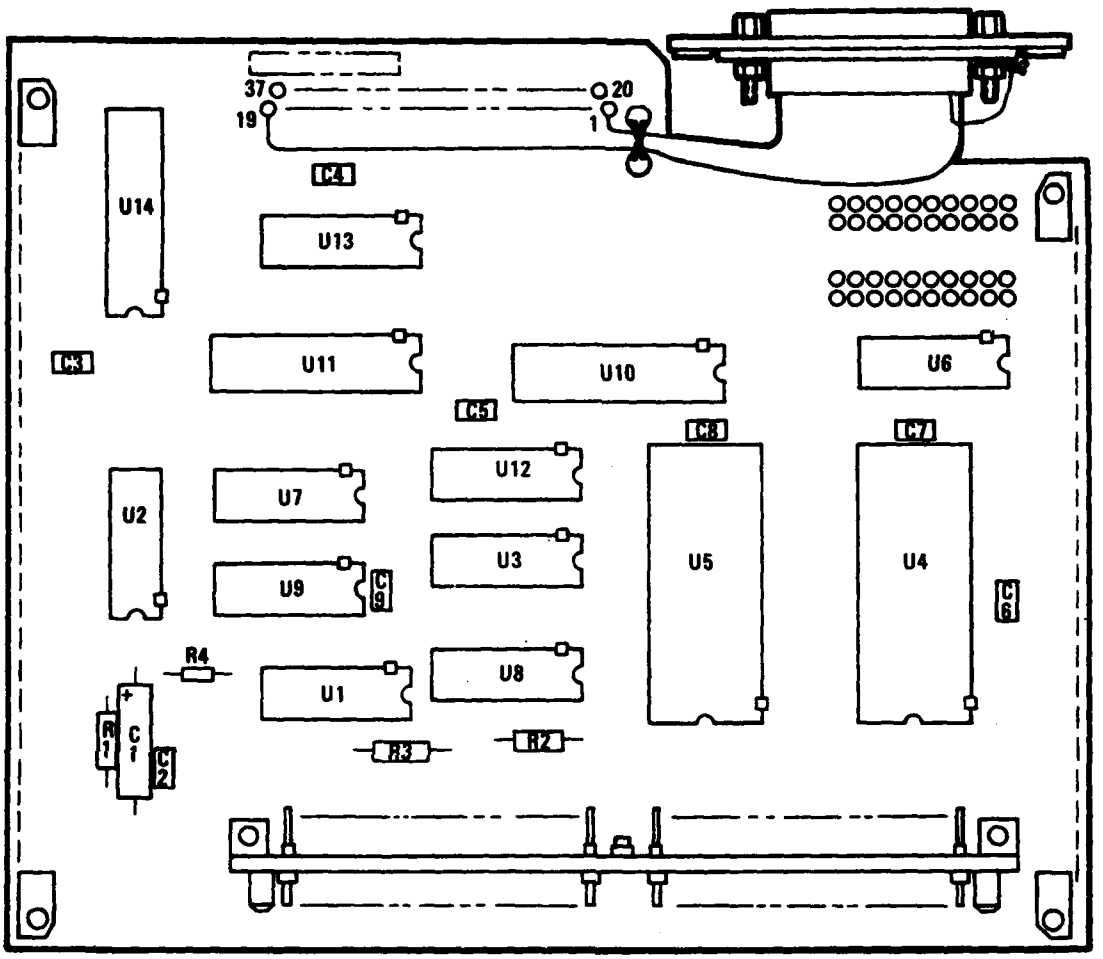


Figure 39. Top Level Assembly Drawing of DMA

MIL-STD-1553B

The 1553B card is the most densely packaged I/O card as can be seen from the assembly drawing of Figure 40. The card contains a number of flatpacks, which in a production atmosphere is undesirable because of assembly cost. The 1553B card, as implemented, complies with a now outdated version of MIL-STD-1760.

The card has been functionally tested and is currently being used as the umbilical link between the MGD missile system and the F-4 launch aircraft.

CONCLUSION/RECOMMENDATIONS

The merits of each card will be discussed in this subsection.

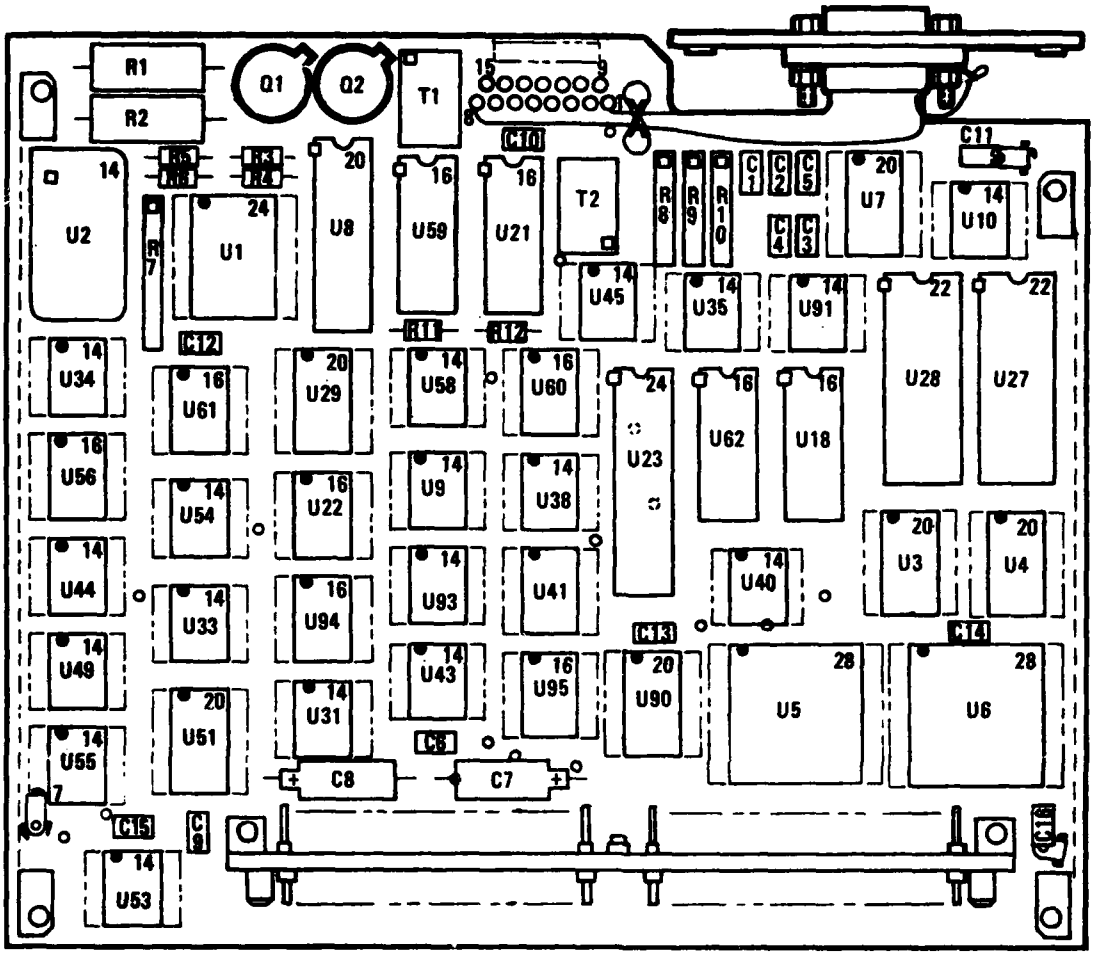


Figure 40. Top Level Assembly Drawing of MIL-STD-1553B

Serial I/O

If the Serial I/O were really to be used in a system other than an embedded computer, it is imperative that the peripheral lines be buffered with a differential drive/receive capability. The noise problem that exists on these lines prevents this card, in its present state, from seriously being considered in a production application.

The internal interface with the CPU should be altered from its present programmed handshake (i.e., CPU interrupt on every word/byte transferred) to a DMA access. The time required to service interrupts for every word in a programmed handshake mode severely limits the throughput capability of the DIS computer. Integration with the low-cost inertial guidance system (LCIGS) demonstrated this fact to the degree that the SIOC on a non-contracted effort

was redesigned to perform an internal DMA function in order to handle the 100 Hz data rates. Also, Draper now agrees, that all I/Os should be DMA in order to prevent overloading the processor.

Parallel I/O

The parallel I/O card could be improved in the same way as the serial card - differential lines and DMA access. The parallel card for the 400 Hz LCIGS Interface was also redesigned to allow that integration take place without a severe systems software overhead burden.

DMA I/O

The DMA card can be improved by adding differential lines to its outputs like the two previous cards. This type of interface should be pursued as a standard. The output can be configured to meet the individual requirements of the peripheral devices (i.e., the BIU and 1553B cards are internally DMA type cards, yet they have different output characteristics).

MIL-STD-1553B

The MIL-STD-1553B card also complies with the original draft version of MIL-STD-1760. Since MIL-STD-1760 has been updated, this card design no longer complies with that specification and should be redesigned. In addition, major improvements by LSI chip manufacturers' products have occurred during the execution of the contract. Many of these new chips are applicable to the MIL-STD-1553B design and should be incorporated in order to reduce parts count, eliminate flatpacks, and reduce total cost. (New LSI devices also would have application in a number of the other DIS hardware cards as well as the DDS designs.) It is conceivable that by using these newer chips, and using the present design experience gained by implementing this card, it is possible to provide a dual redundant channel MIL-STD-1553B single DIS I/O card.

SECTION X

DIS DIAGNOSTIC STATION (DDS)

INTRODUCTION

The DIS Diagnostic Station (DDS) functions as a development station for the DIS computers by providing the utilities necessary to test and debug the programs and identify hardware failures. The DDS also includes the ability to program and erase the PROMs used in the DIS computer.

OBJECTIVE

The DIS diagnostic station is intended to:

a. Debug, edit, and diagnose DIS machine language programs.

b. Link and load DIS programs from the SDC.

c. Diagnose hardware problems in the DIS computers.

The original hardware requirements as stated in the request for proposal were:

a. 16 bit computer

b. Keyboard and display

c. 32K words of memory

d. Dual floppy discs

e. RS-232 modem interface

f. 300 line per minute printer

g. DISMUX interface (BIU)

h. Interfaces for DMA and programmed I/O for up to 8 DIS computers (1553B, serial, and parallel I/O).

Since the DDS was also required to perform as a development station, other requirements were implied. These included the ability to program and erase the PROMs used on the DIS computer as well as a multiuser requirement resulting from the large quantity of software that needed to be developed for the DIS and DDS computers.

DISMUX Interface

The DDS Bus Interface Unit (BIU) is designed to interface the PDP-11 minicomputer to the DISMUX bus. This interface must have the capability of commanding, receiving, and responding to transmissions on the DISMUX bus from

any other BIU address on the bus. The interface must be compatible with the PDP-11 UNIBUS architecture, and be fast enough to respond to the round robin passing protocol (RRPP) of the DISMUX bus. It must also have a self-test capability to ensure the integrity of the DDS.

DMA Interface

The objective of the DMA card was to provide the DDS with a means of communicating with the DIS through its DMA card. It was to attain a minimum speed of 500 thousand transfers per second (Reference Specification No. CS64-32100). It was also required to perform a programmed self-test.

Parallel Interface

The objective of the parallel board was to provide the DDS with a means of communicating with the DIS through its parallel board. It was to be capable of operating at a minimum of 250 thousand word/byte transfers per second (Reference Specification No. CS64-32100). It was also required to perform a programmed self-test.

Serial Interface

The DDS serial interface board provides a means for exchanging 8 or 16 bits of data between the DIS serial I/O card (SIOC) and the DEC PDP 11/34 UNIBUS (DDS) (Reference Specification No. CS64-32100).

1553B Interface

The DDS 1553B interface was designed to provide a 1553 Control Terminal function for the PDP-11. This interface could then communicate with the DIS 1553B remote terminal card. The card was designed to conform to all required aspects of MIL-STD-1553B. Mode codes were not implemented as allowed within the option of the specification.

APPROACH

The direct and implied requirements led to the selection of the DEC PDP 11/34 minicomputer with 128K words of memory, dual cartridge disks, and the peripherals listed in the preceding subsection running under the RSX-11M multitasking operating system.

Five card types were designed to interface to the DIS; they are the BIU, 1553, DMA, serial, and parallel cards. They all reside on the PDP-11 UNIBUS and all but the serial and parallel cards use DMA to transfer data to the DIS computers. These cards provide high speed data transfers to and from the DIS computers to verify and diagnose DIS I/O interface cards.

DISMUX Interfaces

PDP-11/34 UNIBUS Interfacing

The UNIBUS architecture of the 11/34 allows for fast transmittal of data between a peripheral device and the CPU memory by direct memory access (DMA). The DDS BIU was designed to use DMA to transfer messages between the DISMUX bus and 11/34 memory so that it can keep up with the one megabit data rate of the bus.

The BIU in the DDS reacts to stimulus on the UNIBUS in either a bus master or a slave mode. The slave mode is used when the BIU DMA functions are being initialized, or when card status is requested by the CPU. The bus master mode is used when the BIU has been set up to either receive a message from or transfer a message to the DISMUX bus. The BIU takes control of the UNIBUS in this mode, selecting the memory locations it will read from or write to in order to complete the transfer of data.

Proper Manchester encoding and decoding of messages must be accomplished by the BIU. For this task, the Harris Semiconductor part HD-15530 is used (see Figure 41). The Harris chip performs encoding of bit serial data into the Manchester format, including sync and parity information. It also has an independent decoding section, which decodes the Manchester format into synchronous bit serial data. Parity and sync status are also given on separate outputs.

Response Time

To ensure correct operation of the BIU within DISMUX bus timing requirements, it was determined that some form of intelligence was necessary to coordinate end-of-transmission (EOT) recognition, message transmission, message reception, UNIBUS DMA, and the UNIBUS slave function. A microprogrammed sequencer supplies the necessary speed and intelligence while adding flexibility to the design of the BIU, and was therefore chosen for this task.

Self-Test

Self-test must transmit a series of command and data words on the DISMUX bus (with no bus monitors attached) and receive those words using the DMA functions, verifying the sync and parity values of the data transmitted are properly received. In keeping with Digital Equipment Corporation (DEC) software driven formats, the error indication is in Bit 15 of the status register on the BIU.

DMA Interface

Due to the nonstandard handshake of the DIS DMA board, it was decided to do a custom design as opposed to procuring a DR11-B from DEC. To allow the

DDS and DIS computers to operate with a reasonable distance (15 feet) between them, the DDS DMA board was designed with differential TTL drivers and receivers on outputs and inputs. These differential signals are translated to DIS DMA card LSTTL signals at a buffer box located within 2 feet of the DIS computer.

To avoid the problems encountered on the DDS parallel board due to asynchronous operation, the DMA board is synchronous and uses a microprogrammed state sequencer. The sequencer controls a pair of 2940 DMA address generators, the non processor request (NPR) cycle and the bus request (BR) signals.

It was decided to have the self-test feature controlled by the PDP 11/34 rather than by the state sequencer to reduce parts count. Self-test was implemented by fetching a word from memory and storing it in the output buffer while changing the address register such that the wrapped around word would appear in a different memory location. The word was then DMA'd and verified. Errors were simulated also resulting in testing approximately 95 percent of the board.

Parallel Interface

It was decided that, due to the non-standard handshake of the DIS I/O boards, a custom design (as opposed to a Digital Equipment Corporation DR11-C Parallel Interface Board) was adopted. The approach used to solve the communication over distance problem for DMA was used on the DDS parallel board. Differential signals at the DDS are translated to and from LSTTL signals at a buffer box located near the DIS computer.

The self-test requirement for the board was implemented by wrapping a data word from the output buffer back around to the input buffer. Some extra hardware and a considerable amount of software were required to include the self-test feature. By simulating errors in addition to checking the data path, approximately 95 percent of the board is tested.

Serial Interface

The DDS serial board has four registers used to communicate between the serial board and the user (programmer). These registers are control/status, data buffer, interrupt vector, and diagnostic data.

The control/status allows the user to examine the status of the board. Errors detected are transmission errors, parity errors, bad clock errors, and time out errors. The status register also tells the user when data can be transmitted or when received data is ready.

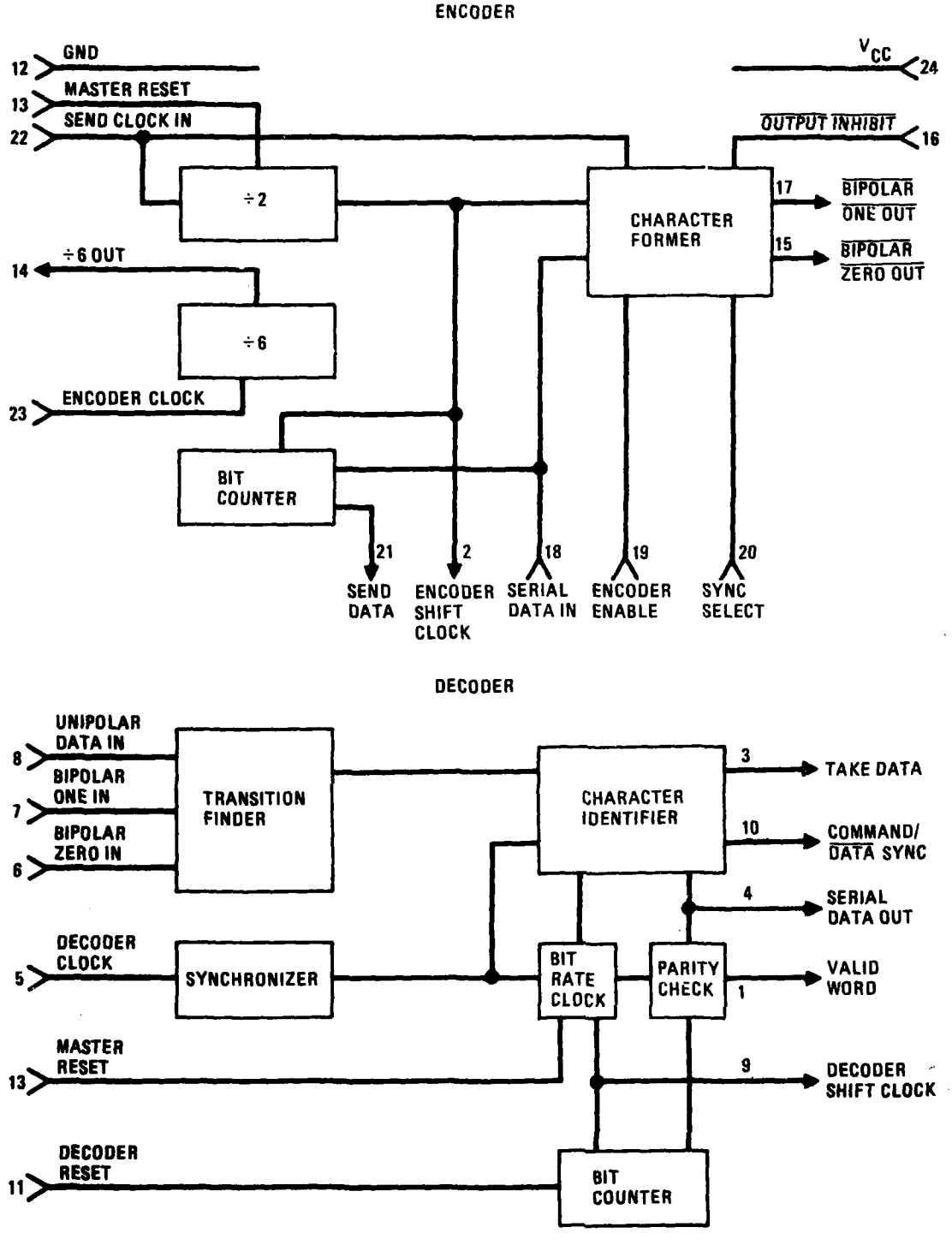


Figure 41. Harris HD-15530 Encoder/Decoder

1553 Interface

The major areas of design work in this interface task were providing a proper PDP-11 UNIBUS interface, ensuring compliance to 1553B, and providing a self-test capability for diagnostic purposes.

Direct memory access (DMA) was chosen as the best means of implementing this interface function on the UNIBUS. DMA allows the processor to continue its task execution while the interface card transmits or receives data. It frees the computer from having to initiate the transfer of every word of data.

RESULTS

All of the requirements for the DDS were met or exceeded; however, as the DDS and DIS were used together several new requirements were suggested to improve the performance or utility of the DDS. They are:

a. Still larger mass storage

b. Faster processor to allow the DDS to handle more I/O

c. More remote terminals to enable more users access to the DDS

d. Functional improvements to the interface cards

e. Provide a low-cost version of the DDA

DISMUX Interface

A block diagram of the DDS BIU (Figure 42) shows the functions implemented on the card to perform DISMUX interfacing. The Advanced Micro Devices (AMD) 2910 microprogram sequencer forms the controlling block of intelligence on the BIU card. Together with the DMA address generator it controls all bus master functions of the UNIBUS, as well as the transmission and correct reception of Manchester encoded messages. All other circuits are support functions, which either perform signal translation (such as the UNIBUS driver-receiver circuits) or give test indication or commands to the sequencer (such as the BOM/EOT circuitry).

Transmission of Data from the PDP-11/34 to the DISMUX Bus

Programming the DDS BIU for transmission of messages consists of preparing a message buffer in memory with the correct message format (BOM, 32 words of data (maximum), EOT). The DMA address register is loaded with the address of the first word of the memory buffer, the word count register is loaded with the number of words to be transmitted (including BOM and EOT) and then the transmit bit (Bit 15) of the command/status register is set. The BIU will wait for its turn in the round robin passing protocol of the DISMUX bus, then transmit the message with proper sync and parity, using DMA to retrieve the

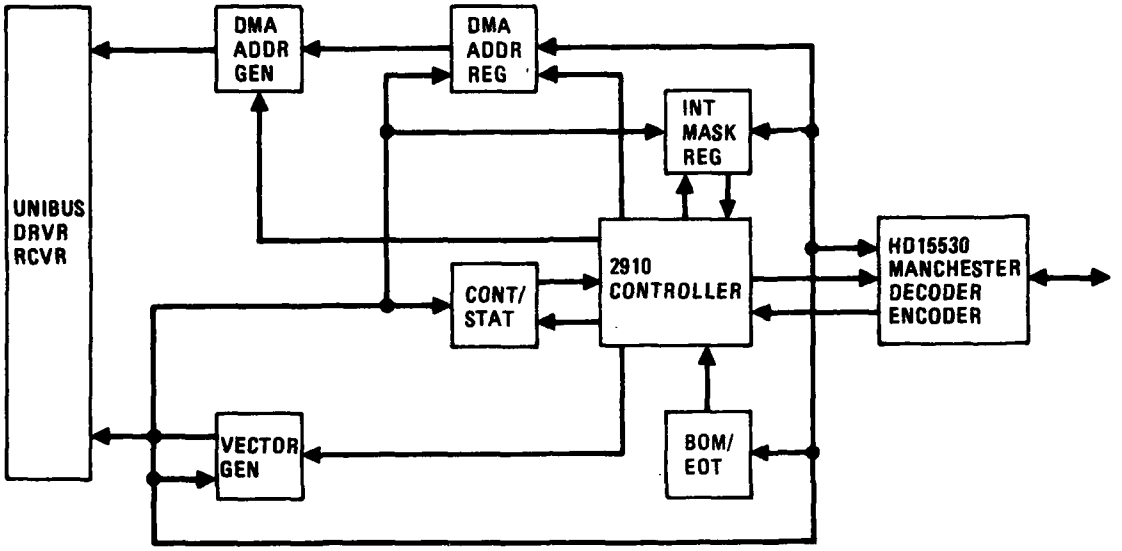


Figure 42. Block Diagram of DDS BIU

message from memory. Upon completion of the transmission, the BIU will interrupt the PDP-11/34 with the DDS interrupt vector to indicate completion of the transmission.

Reception of Messages from the DISMUX Bus

A user desiring to receive a message from the DISMUX bus must first clear the interrupt mask data bit associated with that message, informing the sequencer that that BIU identification and message type are to be accepted and transferred to PDP-11 memory. In addition, the user must load the starting address of the receiving memory buffer in the DMA address register associated with the BIU identification of the message. The BIU, which is continually monitoring the DISMUX bus activity, will receive the message's BOM, compare it with its associated interrupt mask bit, and, if the bit is clear, DMA the message into the receiving memory buffer. An interrupt vector associated with the BIU identification is used to interrupt the processor upon completion of the reception.

The first word in the receiving memory buffer is the word count of the number of words in the buffer (including the count word). If the word count is not three more than the word count in the BOM of the message, an error has occurred. Bit 15 of the command/status register indicates a parity, sync, or incomplete transmission error.

DMA Interface

The DMA board was fabricated, debugged, and installed with no major problems. It ran at the specified speed of 500 thousand transfers per second and could

be extended to 1 million transfers per second with a special miniprogram.

Parallel Interface

The DDS parallel board was fabricated, debugged and installed with no major problems. It was found that the board is significantly faster than the specified 250 thousand words per second; however, the PDP 11/34 cannot stimulate it fast enough to exceed specificaions.

1553 Interface

The DDS 1553 has an AMD 2910 microprogram sequencer to control Bus Master DMA functions on the UNIBUS and to ensure correct 1553 bus transfers.

A bus transfer includes one or more messages transmitted on the bus. The control/status register is loaded with the number of status words to be received. This gives the card an indication of successful completion of a message transfer, which initiates an interrupt to the computer.

CONCLUSION/RECOMMENDATIONS

If future versions of the DDS or similar equipment are designed, several important observations have been made. They are:

a. An effort should be made to keep the system as simple as possible by separating the development and test functions.

b. Specify interfaces that can be implemented using existing off-the-shelf hardware, i.e., a serial interface compatible with existing LSI UARTS instead of an arbitrary format.

c. While the DIS computer requirements were very specific, the DDS requirements were poorly or incompletely specified. The resulting system is more complex than necessary in some areas while painfully inadequate in others.

DMA Interface

Several changes should be made to the DMA Interface card. They are:

a. The state sequencer should be changed to jump to any location in the control store rather than every fourth location.

b. The sequencer should be lengthened to enable it to do self-test.

c. All control register bits should be improved to enable it to do self-test. This means adding another set of registers.

d. The 2940 address generators should be replaced by register/counter chips.

Parallel Interface

Several changes should be made to the Parallel Interface Card. They are:

a. External interrupts should be latched whether or not interrupts were enabled.

b. The entire board should be synchronous (with an onboard clock).

c. A simple state sequencer would be used to generate the bus request signals on the UNIBUS.

d. All control register bits should be mirrored in the status register even if it means adding another set of registers.

Serial Interface

Change the SIOC to a standard handshake either synchronous or asynchronous. With this, it would be possible to use off-the-shelf UARTs. This would require less hardware and would interface to more devices and peripherals.

DISMUX Interface

Interaction of the BIU hardware with the PDP-11 system software, RSX-11M, causes a few problems. RSX-11M is a multi-tasking operating system that demands all Input/Outputs be done through the system rather than through the user program. This causes a reduction in the speed with which the BIU can react to messages on the DISMUX bus. In addition, the software drivers for the BIU are complex and less flexible than is desired for a real-time diagnostics computer.

A solution to this problem could be the rewriting of the drivers and application programs to operate under the real-time system environment of RT-11. The effect on application software would be insignificant and the increase in DISMUX response time would allow more useful monitoring of the bus. RT-11 also has a multi-user version, but the effect of adding multiple users is to slow processor response to outside stimulus such as the DISMUX bus monitoring. It should be noted that the hardware remains the same regardless of which operating system is installed.

One recommendation, therefore, is to use RSX-11M for program development in the multi-user mode, then to switch operating systems to RT-11 for system monitoring and debug. The disadvantage is that only one user can use the computer at a time to debug a DIS system.

Another problem with the BIU is basically programmatic in nature. The DDS BIU was designed to the original specifications of the DISMUX bus defined at the beginning of the contract. Specifically, the BOM word contained five bits of BIU identification, five bits of message identification, and five bits of word

count. The EOT contains unused bits and the BIU identification. Later a decision was made for followup programs to use the five BIU ID bits as additional message ID bits, thus expanding the message types from 32 to 1024. The problem this creates in the DDS BIU is that the BIU ID in the BOM is used to point to a DMA receiving memory buffer (one of 32), and the BIU ID in the EOT is used to create an interrupt vector to point to that memory buffer. Unless the bit fields are the same value (the BOM and EOT BIU ID), the vector will have no correlation to the location of the memory buffer, and hence the data will be lost.

A solution to this problem is to set all DMA receiving memory buffer addresses to the same value, i.e., creating only one buffer for all received data. Under the current constraints of the software driver, this solution does not slow down or diminish the capabilities of the system. In this manner, all interrupt vectors will point to the same buffer, and no data will be lost.

1553B Interface

Two recommendations would be to make the interface MIL-STD-1553B compatible, and to provide a listen only mode to monitor bus traffic when there is another controller on the bus. The latter recommendation is merely a change to the microprogram sequencer's memory.

The 1553B interface has operated successfully with the DIS breadboard, brassboard, and delivered computers.

SECTION XI

IIAS-LCIGS INTEGRATION

INTRODUCTION

An Integrated Inertial Avionics System (IIAS) was developed to demonstrate the DIS federated computer system operating with the DISMUX bus. This system consisted of five IDS flight computers, a backup autopilot (BAP) computer, and a carrier aircraft interface processor (IP) computer along with their associated software. In addition, a simulation integration laboratory (SIL) was developed to drive, monitor, and respond to the flight computers in a closed loop mode. The five DIS computers served the five functions of Supervisor (SUP), Inertial Navigation Reference (INR), Guidance and Navigation (GAN), Digital Autopilot (DAP), and Vehicle Interface Processor (VIP). All of the application software developed for these DIS computers was programmed in JOVIAL J73 HOL. The BAP computer was programmed in assembly language and the IP was programmed in FORTRAN.

The hardware integration of the low-cost inertial guidance subsystem (LCIGS) and the digital integrating subsystem (DIA) is the first step in demonstrating the applicability of LCIGS and DIS in a future midcourse guidance demonstration (MGD) flight test program.

OBJECTIVE

The objective of these integration tasks was to develop a working computer system, in terms of both hardware and software that was applicable to missile avionics. All DIS computers must share information and results in an organized manner in order to be successful.

The LCIGS/DIS hardware integration task was to electrically transfer 400 Hz LCIGS parallel data to a DIS computer and electrically transfer 100 Hz LCIGS serial data to a different DIS computer. To meet the objective, several lower level objectives must be successfully completed. They were:

a. PSE Installation/Checkout

b. LCIGS Operational Verification

c. LCIGS-DIS Physical Integration

d. LCIGS-DIS Parallel Verification

e. LCIGS-DIS Serial Verification

APPROACH

The computer system was built up to five computers and all subsystem elements by using a series of small integration steps, or "builds." The first build, for example, used a single supervisor (SUP) computer. All software and hardware elements, interfaces, etc., were successfully integrated. The second build had a second computer, with its associated elements, and both computers were exercised to validate appropriate message transfer. The "build" process continued until the entire system was completed.

Flight Hardware

The flight hardware was organized for the five DIS computers as listed in Table 16.

TABLE 16. FLIGHT HARDWARE FOR DIS COMPUTERS

Computer	Memory Allocation		Class	I/O Card Complement
	Instruction	Data		
SUP	8K	8K	I	BIU 1553 MK117 RCDR
INR	12K	4K	II	BIU LSIO DISC RCDR
GAN	28K	28K	I	BIU - - RCDR
DAP	12K	4K	II	BIU LPIO RADAR RCDR
VIP	28K	28K	I	BIU D/A A/D DISC

I	-	Class I CPU (350 KOPS)
II	-	Class II (500 KOPS)
BIU	-	DISMUX Bus Interface Unit Card
1553	-	1553B Bus I/O card
MK117	-	MK117 Bus I/O card
RCDR	-	Flight data recorder I/O card
LSIO	-	LCIGS Serial I/O card
LPIO	-	LCIGS Parallel I/O card
DISC	-	Discrete I/O card
RADAR	-	RADAR I/O card
D/A	-	Digital to Analog output card
A/D	-	Analog to Digital input card

The BIU card in the SUP computer was programmed to provide the DISMUX exception handling function, i.e., to initiate the round robin sequence and to maintain it in the event of other BIU malfunctions. The round robin sequence chosen is in the order of the computers as shown in Table 16. The BIU IDs were chosen so that there is at least a two bit difference between any two computers, minimizing the chance for two computers trying to gain access to the bus simultaneously if one should mis-read an EOT. The individual BIUs were programmed to read all messages needed for a flight environment. In addition, the remaining ID mailboxes were programmed as spares in an attempt to accommodate new messages without requiring the burning of new BIU PROMs.

PSE Installation/Checkout

To perform calibration, acceptance testing, and field maintenance of the LCIGS, LSI designed and built an item of Peculiar Support Equipment (PSE). The PSE was delivered, installed, and checked out at GDC without major problems. Additionally, the PSE diagnostics were successfully exercised.

LCIGS Operational Verification

After the PSE was installed and integration to the Gyro Table Electronics was completed, static (TCAL2) and dynamic (TDYN2) calibrations were performed. The unit successfully demonstrated that it was capable of passing both static and dynamic calibration verification.

LCIGS-DIS Parallel Interface

Originally a parallel input-output card was to be used to interface the LCIGS to the DIS computer. However, after evaluating the load on the operating system it was determined that computer efficiency would be greatly improved if a DMA parallel interface were used instead of parallel I/O under I/O control. Additionally, to improve operation further, a countdown word counter was added to effectively convert the LCIGS 400 Hz data to 50 Hz data. The counter was designed to count eight transfers of nine byte blocks of LCIGS 400 Hz data. When eight block transfers were received, a single interrupt was generated to the DIS computer by the LPIO card. Conceptually, a toggle flip-flop was added to the design to provide a "ping-pong" starting address for input block transfers. This allows the operating system and task software access to a received block of data upon input, while receiving new data into a contiguous "mailbox." The toggle flip-flop alternately changes the least significant bit of the block ID upon completion of each eight block input transfer, thus pointing to two starting addresses in an alternating fashion.

Implementation of the above design suggestions was accomplished by modifying an existing DIS DMA card. Designation for this card is LCIGS Parallel I/O (LPIO).

Data transfer from the LCIGS to the DIS via the LPIO card was successfully completed. Hardware problems were not encountered. A software handler was written that would fill two mailboxes and list them on the line printer.

As a point of information, the LCIGS data is sent as bytes and is placed into the mailboxes on word boundaries. The user of the input data must realize that every other byte of his input buffer is valid. A DMA driver has not been written yet to transfer the data to the application task in the form of a byte array.

LCIGS-DIS Serial Interface

Originally a serial input-output card was to be used to interface the LCIGS to the DIS computer. However, similar to the LPIO, it was determined that computer efficiency would be greatly improved if DMA communications was implemented. Additionally, to improve operation further, logic was added that would provide a toggling mailbox address so that received data could alternately be transferred to two consecutive mailbox addresses. Each block of received data at the serial interface is composed of 12 bytes. Conceptually, similar to the LPIO, a toggle flip-flop was added to the design to provide a "ping-pong" starting address for input block transfers. This allows the operating system and task software access to a received block of data upon interrupt, while still receiving new data into a contiguous "mailbox." The toggle flip-flop alternately changes the least significant bit of the block ID upon completion of each block input transfer, thus pointing to two starting addresses in an alternating fashion. Additionally, parity logic was added to indicate whether a parity error exists somewhere in an entire block of transferred data.

The above design recommendations were implemented by modifying an existing serial input-output card. Data transfer from the LCIGS to the DIS via the LSIO card was attempted. An interface anomaly was encountered on the $\overline{\text{CRX}}$ line, which required an additional design change to reduce noise and ringing. The interface was then successfully demonstrated.

Flight Software

The flight software was partitioned among the five DIS computers as follows.

SUP Computer

The SUP computer contained in its ROM the 1553B bootstrap software as well as DISMUX bootstrap software (which all the other DIS computers contained). This allowed SUP to be downloaded from the IP via the 1553B as well as from the DDS via the DISMUX. SUP also contained 1553B/DISMUX and 1553B/MK117 transparency functions that allowed the IP to communicate with any of the DIS computers or with BAP for the purpose of downloading them or sending them data. SUP has the task of receiving status data from the DIS computers and

using this data to formulate a DIS satisfied flag for transmission to BAP. The absence or mis-configuration of this flag could be construed as a signal for BAP to assume control of the flight control system.

Two other functions of SUP were telemetry and DISMUX data recording. SUP's BIU was programmed to accept all DISMUX messages. These messages were sent to the flight recorder as received. In addition, certain messages were sent to BAP for incorporation into the telemetry stream along with BAP telemetry data.

INR Computer

The INR computer was primarily concerned with the strapdown navigation algorithm. This is a high processing rate function and required INR to be a Class II machine. Another function of INR is to handle the output discretes that would not fit on the discrete card in VIP. The functional requirements for these discretes still largely remained in VIP, which generated a DISMUX message containing the discrete and INR issued the discrete via its output card.

GAN Computer

The GAN computer contains three major functions: guidance, navigation, and the transfer alignment Kalman filter.

The guidance algorithm provided waypoint steering and cycled at two frequencies. The major guidance cycle was 2 Hz with a "fast steer" cycle at 10 Hz. The output commands of the guidance program were sent to DAP via the DISMUX.

The navigation software cycled at 10 Hz. It received the strapdown integrated velocities and angles from INR and transformed these into inertial position and velocity. This data was then used by the Kalman filter during the transfer alignment phase and by guidance during the simulated flight phase.

The Kalman filter is an 18 state filter with only 15 states in process at any one time. Following transfer alignment, two g-sensitive drifts and the X accelerometer scale factor states are dropped and three position states added.

DAP Computer

The DAP computer processed the LCIGS parallel 400 Hz data for vehicle stability and control. The 400 Hz delta velocities and angles were input into a digitally derived rate (DDR) filter to obtain vehicle body rates and accelerations. This data was used to stabilize the vehicle in the autopilot function. Guidance commands were received from GAN via the DISMUX. The air data needed for vehicle control was received from the VIP computer's A/D card via the DISMUX. DAP also was the recipient of the RADAR data, which was used only for information and not for vehicle control.

VIP Computer

The VIP computer served as the interface between the DIS computers and the vehicle. It contained the software necessary to interface with the I/O cards and the software to sequence the vehicle flight functions. The override processor software also resides in VIP. This software decodes the commands from the manual override system and sends appropriate data to GAN so that the proper vehicle maneuvers are followed.

BAP Computer

The BAP computer contains the software for a backup autopilot sequencer and override processor function. In addition, it formats the telemetry for the software portion of the pulse code modulation (PCM) system. BAP continuously (10 Hz) monitors the receipt of the DIS satisfied flag. If this flag is not present or configured properly for 0.2 second, BAP will assume control of the vehicle. BAP does not have a guidanced function and therefore will fly the vehicle on a straight and level trajectory until commanded otherwise by the override system, or until control is switched back to the primary (DIS) system.

Utility Software

A full set of utility software was coded in JOVIAL J73. This included the usual trigonometric routines and their inverses, square root, and vector routines. These utilities were written for floating and fixed point application, and most of the fixed point routines were done in both single and double precision.

Simulation Hardware

The simulation hardware comprised a set of four Interdata 8/32 computers driving and responding to the DIS computers, the IP, and the BAP computer through appropriate I/O devices. This suite of hardware allowed a complete closed loop operation and checkout of the flight software. Various output devices, tape recorder, printer and CRT, allowed data retrieval of flight software events both in real time and for post-run analysis. One of the more important data retrieval items was the ability to extract and record all of the DISMUX traffic and display it in engineering units.

Simulation Software

The simulation software provided a complete simulation capability of the flight vehicle, carrier aircraft, GPS X-set, LCIGS platform, and other appropriate devices. The software functions in the four Interdata computers were divided as follows:

Computer 1 Real-time control, data retrieval, and output

Computer 2 LCIGS, missile, and carrier aircraft six degree-of-freedom (DOF) simulation

Computer 3 X-set simulation

Computer 4 Graphics display

Software Development

System Design

Software was developed by first documenting the requirements for each flight computer in the form of engineering design requirements bulletins (DRBs) and then B5 and C5 specifications. A system level (type A) specification governed the overall flight software requirements and the interface between the DIS computers. Software design was accomplished using the NASA developed software design and documentation language (SDDL) system. This allows the design of software in a language-independent format at a high level. The design was then implemented in JOVIAL J73 using the DIS JOVIAL compiler, which was resident on the IBM 370 computer. The interface to the IBM, and the tool utilized for all software development, was the Harris computer based software engineering system (SES).

Module/Computer Level Checkout

The compiled DIS software was checked at the module level by means of a Z8000 simulator resident on the SES. This simulator utilized a resident Z8000 assembler and FORTRAN linkage for I/O. FORTRAN drivers were used to exercise the individual modules and verify the correctness of the JOVIAL code.

Following the module level checkout, the software was down loaded to the DDS where the DIS assembler and loader resided. The resultant DIS load modules were then loaded directly into the appropriate DIS computers for computer level checkout. This level allowed the first interface of the DIS operating system with the application software, as the operating system was not used in the simulator mode.

System Level Checkout

Following computer level checkout the software was exercised in the SIL for system level checkout. This checkout proceeded in four software builds. The first build was primarily to test the IP/SUP download interface. The second build tested all other system interfaces. In this build all computer I/O was exercised and all DISMUX messages were generated and received. The software functions to generate these messages were primarily stubs. The third build was to verify the transfer alignment function, and the fourth build was to be the complete vehicle functional capability.

System Storage/Duty Cycle

The present view of DIS software storage and duty cycle for build "four" is shown in Table 17. The duty cycle numbers reflect the fact that INR and DAP are Class II machines.

TABLE 17. BUILD FOUR DIS SOFTWARE STORAGE AND DUTY CYCLE

	Memory		
Computer	Instructions	Data	Duty Cycle
SUP	6,721	4,390	65.3
INR	6,694	2,171	81.2
GAN	23,837	14,952	70.5
DAP	10,770	3,633	54.9
VIP	10,322	2,421	44.7

RESULTS

The final proof as to the validity of these integration tasks will take place on the first MGD test flight schedule in October of 1981.

It appears that the only problems encountered on the IIAS and LCIGS tasks were the general problems that can be expected in any new computer system where the airborne hardware, software, support hardware, and support software are all maturing at the same time.

CONCLUSION/RECOMMENDATIONS

It is appropriate to comment on some potential problems with regard to the LCIGS integration tasks. These are listed below:

Hardware Reset

To properly utilize the hardware reset function, the states of the control lines, before, during, and after assertion must be defined and agreed to by both LCIGS and DIS personnel. Presently the control lines between the LCIGS and DIS are asserted when hardware reset is active and are de-asserted when hardware reset is deactivated, indicating that a valid data set can be placed into memory. This can cause erroneous data transfers to the DIS computers.

Initial Communication Establishment

The LCIGS, GAN, and INR computers are located on the same 28-volt power bus on the MGD missile. When power is first applied, each unit has a software initialization sequence to perform before being capable of operating properly. Definition of the minimum time before LCIGS-DIS communication attempts occur must be made. The intention is to ensure that LCIGS does not attempt communications for approximately 0.25 to 0.5 second after power application. This will allow the DIS operating systems to initialize the LPIO and LSIO cards before LCIGS-DIS communication starts.

LSIO Output Buffer

Future designs of the $\overline{\text{CRX}}$ and $\overline{\text{CTX}}$ interface between LSIO and LCIGS should not use the LS240 device.

TTL Interface

It was also clear that the original contract agreement to use single ended TTL devices for the high data rates between guidance packages was a mistake. Significant time and money were expended to recover from this error.

SECTION XII

FAST FLOATING POINT TRADE STUDY

INTRODUCTION

The selection of a particular hardware implementation for fast floating point within the DIS computer was determined by the execution speed, hardware and software impacts, and availability of that implementation. This study has determined that the Zilog extended processing unit (EPU) is the most desirable candidate if maximum DIS legacy is to be maintained. The EPU offers the cleanest hardware solution with a moderate impact on the DIS assembler, linking loader, operating system, and compiler. For future DIS applications, the EPU can be considered a viable candidate.

For near-term applications, or applications where the lack of the EPU may be detrimental to the success of the program, and where DIS legacy is to be maintained, a bit slice emulation of the Z8000 incorporating hardware fast floating point execution is the next best candidate. It offers a low risk implementation at a moderate cost of hardware design. A majority of the current DIS system can be retained if this candidate is chosen.

The Intel coprocessing candidate containing the 8086 CPU and 8087 coprocessing element is another solution to hardware floating point execution. It would require a major change to the DIS hardware and software systems. It offers a viable solution to the execution of fast floating point and should be considered in applications where retaining the maximum DIS legacy is not a requirement. Intel is on the verge of committing some of its resources to the military market and has already identified plans for support of Air Force recognized high order languages, second sourcing, and licensing agreements.

The EPU, bit slice emulation, and Intel 8086/8087 candidates are the only approaches evaluated that offer execution speeds considered adequate for airborne applications. The other candidates evaluated did not offer comparable speeds or straightforward hardware interfaces to the DIS computer.

OBJECTIVE

General purpose microcomputers like the Z8000 based DIS computers require extensive support software to perform fast floating point arithmetic for process control, navigation, and Kalman filters. Some applications require an accurate and precise format at a relatively moderate processing speed, while other applications require less accuracy but at a fast execution speed. Since the current DIS computer provides floating point execution via a 16-bit exponent and a 32-bit mantissa in software, it is worthwhile considering an alternative hardware approach to fast floating point execution. Commercially available "number

cruncher" chips and arithmetic processing chips, as well as bit slice arithmetic processing units lend themselves to meeting these needs.

Selection of a particular hardware implementation involves careful evaluation of cost, speed, size, power consumption, ease of interfacing with existing hardware, and minimum impact on existing software. The purpose of this study was to determine the best fast floating point hardware solution for the DIS computer.

In choosing a hardware implementation of hardware floating point, the first and probably most obvious concern is the interfacing of the floating point hardware to the existing microprocessor architecture. One such interface approach is shown in Figure 43. In this configuration, the microcomputer will handle instruction and data transfers to and from the floating point hardware (FPH) as shown in the flow chart of Figure 44. The microcomputer polls the FPH ready status for completion of an instruction. The CPU then removes the result of the previous operation and supplies the next instruction to the FPH. The polling can be replaced with an interrupt upon completion of the FPH, freeing the CPU to perform parallel tasks during FPH operation. Either of these approaches is relatively simple and low-cost; however, the speed of execution is penalized by the time required to exchange data and instructions between the FPH and the CPU as well as the input/output card calling sequence and timing protocols.

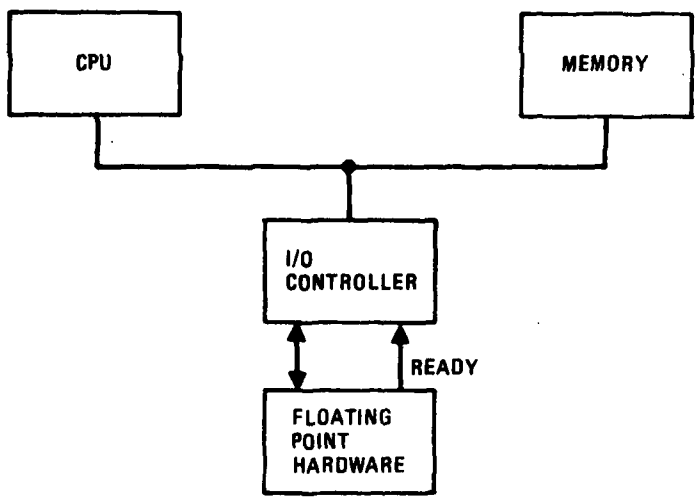


Figure 43. Basic Interface of Floating Point Hardware to the Microcomputer

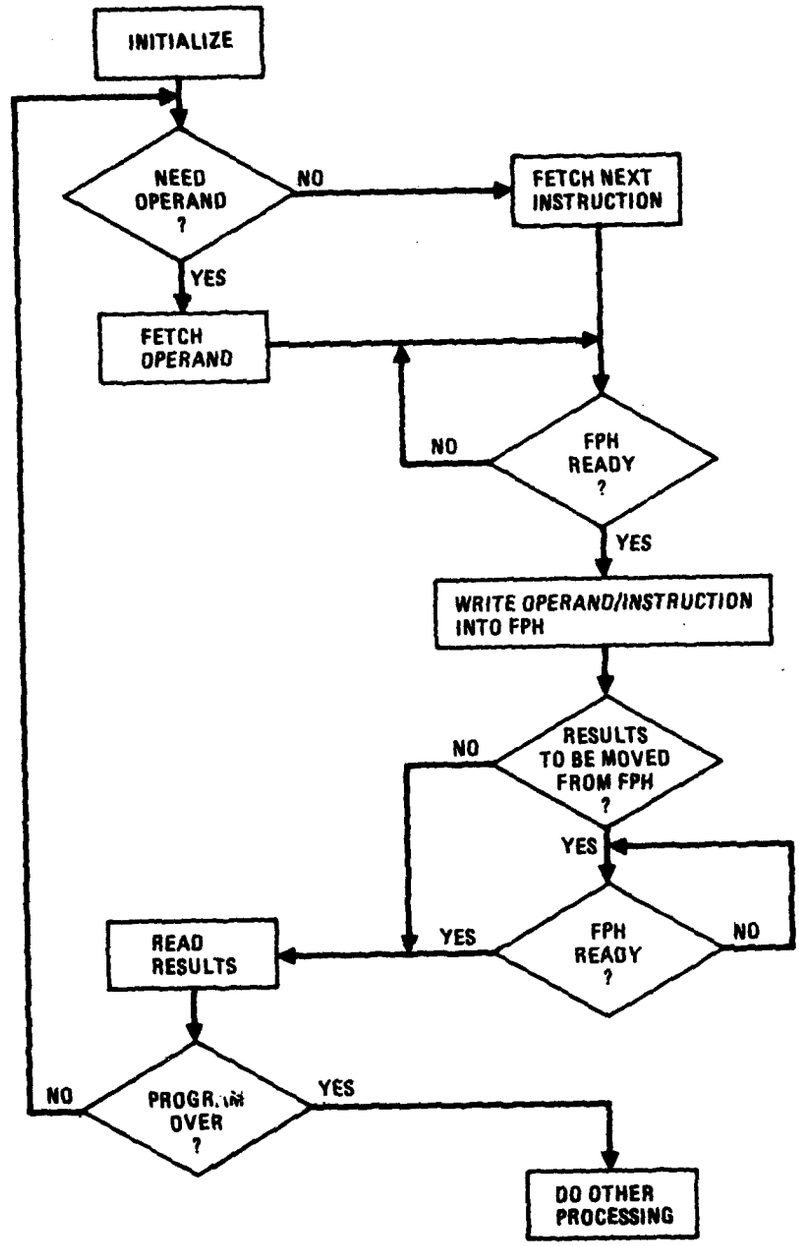


Figure 44. Floating Point Hardware (FPH) Access in the I/O Mode

If the floating point algorithm were fixed (and for our application it is not), a dedicated set of FPH with the routine stored on ROM could be used. The speed of execution would then be faster.

A more complex interfacing technique that results in a higher speed (because of parallel execution) is shown in Figure 45. This approach requires a separate sequencer/micromachine to transfer instructions and data between the CPU and FPH, along with bus arbitration logic. This multiprocessing mode requires the CPU to transfer data to the FPH local memory, initialize programs in the FPH instruction memory, and trigger execution of the task. Like the previous architecture example, the FPH can signal completion of execution by either interrupting the CPU or via polling the status.

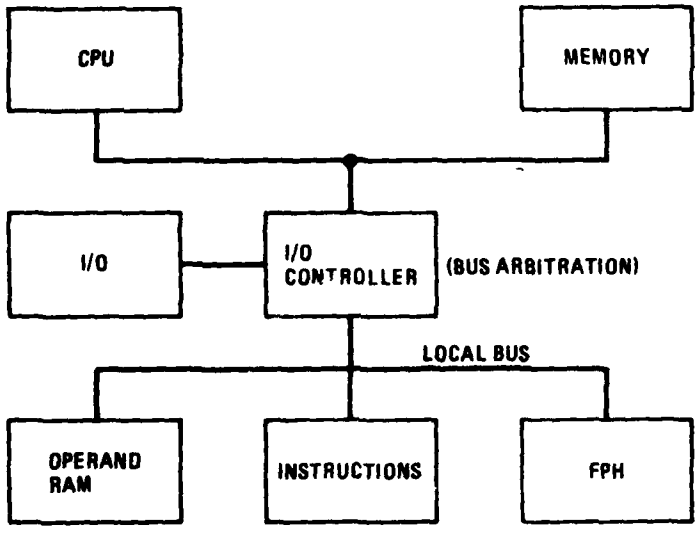


Figure 45. Multiprocessing of CPU and FPH

The hardware floating point implementations presented here possess markedly different architectures. Accordingly, interfaces, programming, and execution speeds will vary with the candidate. We have evaluated all practical hardware options for a hardware floating point implementation by trading off these options against the current software implementation as the baseline. Appropriate criteria, including execution speeds of addition, subtraction, multiplication, division, cost impacts of changes in hardware and software, compiler implications, floating point formats, component availability, and a schedule risk assessment were used to determine the optimum solution.

The baseline software subroutine approach currently provided in the DIS contract executes floating point in the following times:

Add	90 microseconds
Subtract	90 microseconds
Multiply	180 microseconds
Divide	180 microseconds

The hardware floating point hardware options that were evaluated are:

a. Advanced Micro Devices 9511 Arithmetic Processing Unit.

b. Advanced Micro Devices 2903 Bit Slice Arithmetic Logic Unit Add-On.

c. Zilog Extended Processing Unit.

d. Advanced Micro Device 2903 Bit Slice CPU.

e. Intel 8086 CPU with the 8087 Coprocessor Unit.

APPROACH

This section identifies the most optimum implementation for DIS computer fast floating point implementation. To find the optimum implementation, each candidate was compared against various criteria. Each criterion has associated with it a weight value. An important item could have a numerical value of 10 with respect to others, while the least important item would have a value of 1.

Each candidate is given a rating (1 for least performance through 10 for best performance) with respect to each item on the criteria list. A total numerical value assigned to each candidate is equal to the product of each criterion weight times its performance level number added to all criterion products. The candidate with the best performance will therefore have the highest numerical total.

Table 18 is the hardware floating point trade matrix that contains the criteria that were evaluated for each candidate and the results of the study. Each criterion will now be discussed as it applied to each candidate.

Execution

The most optimum hardware fast floating point candidate will require an execution speed approximately 10 times faster than the present software baseline speed. To evaluate the candidates in terms of execution, and other criteria, the execution has been given the highest weight possible because of its importance. Other criteria can have a maximum value of 5. This difference will add the importance of the execution speed.

The software floating point baseline executes fast floating point arithmetic on the average of 150 microseconds. Of all candidates this is the slowest speed and has been given a rating of 1.

TABLE 18. HARDWARE FLOATING POINT TRADE MATRIX

	WT	BASELINE	APU	2903	EPU	BIT SLICE	8086/8087
1. EXECUTION	10	1	2	7	10	10	9
2. HARDWARE IMPACT	5	10	7.5	5.5	10	2.5	3
3. SOFTWARE IMPACT							
A. SYSTEM ENGR	1	10	9	9	9	9	1
B. TECHNICAL/SUBCONTRACT MGT	1	10	9	9	9	9	1
C. OPERATING SYSTEM	3	10	7	7	7	7	1
D. ASSEMBLE/LINK/LOAD/CML	3	10	10	7	7	7	2
E. MATERIAL	1	10	8	4	4	4	1
4. COMPILER IMPACT	5	10	8	7	7	7	1
5. AVAILABILITY							
A. VENDOR HARDWARE	3	10	10	10	1	10	10
B. VENDOR SOFTWARE	3	10	10	10	7	10	7
6. FORMAT	1	10	6	8	8	8	8
TOTAL	36	270	241	265	281	280	181

The AMD 9511 APU has an average execution of 125 microseconds. This includes the best possible execution speeds and the worst possible execution speeds in that implementation. It was given a rating of 2 in this category.

The 2903 Add-On has an average fast floating point execution of 43 microseconds. Its score therefore in terms of execution has been given a rating of 7.

Zilog's EPU execution speeds are projected at this time in the sub-10 microseconds range. Of all candidates evaluated, this potentially has the fastest execution speed of fast floating point. It has, therefore, been given a rating of 10.

The total bit slice Z8002 emulation according to Osborn and Associates has an average execution time of approximately 15 microseconds. This also would meet fast floating point requirements and has also been given a rating of 10.

The Intel 8086/8087 combination has an average execution of approximately 18 microseconds. It has been given a rating of 9; however, its speed is probably adequate and would meet or exceed identified requirements.

Hardware Impact

The hardware impact is considered second in priority to the execution speed and therefore has been given a weight of 5.

The baseline, since it is software dependent only, requires no hardware changes, or new hardware. Therefore, the baseline has scored a 10 in this category.

The 9511 APU would require a change to the present CPU card of approximately 10 integrated circuits (ICs) with the material costs estimated at $100 total. The modification of this card is considered a moderate level task and therefore, together with the parts, the APU has been rated a 7.5.

The 2903 Add-On would require an additional card containing four 2903s and approximately 22 additional ICs. This material cost is estimated at $1000 and the design effort is considered a somewhat difficult task. Because of these factors, the 2903 Add-On has been given a rating of 5.5.

The Zilog EPU requires the addition of a single chip to the CPU card, the EPU itself. The projected cost of this part in quantities downstream is approximately $155. The design task to incorporate this in hardware is considered an easy task and therefore the EPU candidate has been given a rating of 10. Clearly it is the least complex hardware solution of the candidates.

The bit slice emulation would require a complete redesign of the CPU and is projected to have 100 ICs in its design. The material costs of approximately $2K - 5K are considered insignificant. The design task, however, is considered a difficult task and the resulting CPU would be very high in power consumption. The bit slice emulation has therefore been given a rating of 2.5.

Intel's 8086/8087 combination would require a new design of the CPU consisting of approximately 60 ICs and the design task is considered difficult. This hardware combination has scored a 3.

Software Impact

The software impact of these candidates has been broken down into five areas. Each will be discussed at this time.

System Engineering

This category is considered a somewhat insignificant task but one that would exist for overall control of software packages currently being used that may or may not have to be modified. The weight of this category has been set at 1 since its impact is basically manhours and schedule impact.

Since its baseline already exists, system engineering would not be required for new tasks and therefore the baseline has been given the highest grade of a 10.

Four of the remaining five candidates will require softwares modification to some degree either to the assembler, the linker/loader, and/or the control,

monitor and load software packages. Because the relative size of these changes for four of the five candidates is approximately the same, the cost for software engineering for these efforts is also the same. For the APU, the 2903 Add-On, the EPU, and the bit slice emulation, the cost is projected at $5000 for software system engineering. Therefore, all four candidates have received a rating of 9.

The Intel 8086/8087 combination would require major redesign of all software packages currently being provided under the DIS contract. These packages would include the assembler, the linker/loader, and the control monitor and load. Because this effort is considered significant, the software engineering task associated with the redesign is also considered significant and has been estimated at a cost of $50,000. Because of this, the 8086/8087 combination has been assigned a rating of 1.

Technical/Subcontract Management

This task, like system engineering, would interface personnel either in-house and/or through subcontractors, to make sure modifications for the particular software package have been met. It is considered of insignificant schedule impact when compared to other criteria. Therefore it has only been given a weight of 1.

Again, because the baseline exists, its rating is 10 since no changes to the software packages are required.

The same four of five remaining candidates (the APU, the 2903 Add-On, the EPU, and the bit slice emulation) are in the same category of a modest software redesign and therefore their technical/subcontract management task is estimated at $5000. All four of these candidates have therefore been given a rating of 9.

The 8086/8087 combination has been given a cost estimate of $50,000 in a technical/subcontract management area because its software modification is considered significant. Its rating, therefore, is 1.

Operating System

The operating system is more significant of an effort than basic management of the change. It has therefore, been given a weight of 3 when evaluating the candidates.

The baseline, requiring no operating system changes, has received a rating of 10. All remaining candidates except for the Intel candidate would receive a 7 in this category since their modest operating system changes, which include test and documentation, are estimated at $15,000 each.

The 8086/8087 combination will have a significant cost impact to change the operating system and is estimated at $50,000. A rating of 1 has been assigned.

Assembler/Linker/Loader/CML

Like the operating system, these software packages are considered more significant than the management of their change. They have, therefore, been lumped into a single category and given a weight of 3.

The baseline, requiring no modifications in these areas, receives the highest possible rating of 10.

The APU would require no changes to the assembler, linker/loader, or control, monitor, load to implement; therefore, it also received a 10.

The 2903 Add-On, the EPU, and the bit slice emulation each have been given a rating of 7 since it is estimated that changes to the assembler, linker/loader, and control monitor load would cost $15,000 each. In all probability, the EPU option would require no changes to the control monitor and load.

The 8086/8087 candidate, again deemed significantly different from the existing Zilog DIS computer, would require major changes to these software packages and the estimated cost is $45,000. A rating of 2 has been assigned.

Material

Material in this category refers to tapes, disks, disk cartridges, IBM computer time, and other material costs that would be required to implement the software changes described above. This is not considered a significant cost and has been given a weight of 1.

The baseline, requiring no changes, again scores a 10.

The 9511 APU has a material estimate cost of $20,000 to implement the software changes. This $20,000 cost converts to a rating of 8.

The 2903 Add-On, the CPU, and the bit slice emulation all have an estimated material cost in this category of $35,000. They each have received a rating of 4.

The significant change required for the Intel implementation has a material cost estimated at $110,000. This $110,000 is the worst case of the candidates and therefore has been rated a 1.

Compiler Impact

Because of the high cost associated with the DIS compiler, this category has

been given a weight of 5, the highest weight allowed, other than the hardware execution time.

Since the baseline requires no changes, it receives the highest possible rating of 10.

The APU would require some modifications to the compiler. These modifications are required to reflect the APU's single precision format and the redesign and implementation of the floating point library to interface directly to the APU. Its cost is estimated at $30,000 and has been given a rating of 8.

The 2903 Add-On implementation would require a compiler change that is similar to the Zilog EPU and the bit slice emulation. All three compiler change costs are estimated at $70,000. The DIS cross compiler will require a complete redesign of the current floating point implementation. It would also require a change in the current single precision format and, in the case of the EPU, the inclusion of the double precision format. A redesign of the "floating point-related cogeneration" from calls to run-time floating point routine to in-line generation of the EPU instructions would also be required. These three candidates have, therefore, been given a rating of 7.

With respect to the Intel option, the cogenerator and target dependent front end portions of the DIS cross compilers (currently hosted on the IBM 370 and DEC 10 computers) would have to be completely redesigned, coded, acceptance tested, and documented for this option since they are currently designed to take specific advantage of the Z8002 microprocessor. The task is considered significant at a cost of $400,000 and is the worst case compiler impact cost. A rating of 1 has been assigned.

Availability

"Availability" refers to vendor hardware and vendor software support packages used to implement the various candidates. Both the hardware and software packages have been separated in this category and both been given a weight of 3.

Vendor Hardware

The baseline requires no hardware implementation changes and therefore receives a grade of 10.

The APU, the 2903 Add-On, the bit slice emulation, and the 8086/8087 hardware are all available today in some form of commercial and/or military grade. Therefore these four options have been given the highest rating of 10.

The Zilog EPU hardware does not currently exist and is not projected to exist until late 1982 or early 1983. This is considered maximum risk and a rating of 1 has been assigned to the EPU.

Vendor Software

The baseline requires no software from the vendor since it currently exists and therefore has received the highest rating of 10.

Support software for the 9511 APU, the 2903 Add-On, and the bit slice emulation all exists from AMD. These three candidates, therefore, have been given a 10 in this category.

The Zilog EPU software package for verification of software routines will be available the second or third quarter of 1981. This is significant since this software package will emulate the EPU function and allow software verification in absence of the actual EPU chip. This is accomplished in the software package by "trapping" on an EPU instruction and executing the floating point routine in non-real time. This does require a V-step version of the Zilog CPU to operate. The V-step CPU is available and could support this software verification next year. Because the availability of the software package in its final form still does not exist, the EPU has been given a rating of 7.

Although the Intel hardware exists today, the support software package exists only in a preliminary, nonusable form. It is projected to be available the first quarter of 1982 and could support this implementation if it were chosen. It has been given a rating of 7.

Format

This category refers to the actual floating point format that would be used in each of the candidates. It is not considered an important category but is used as a means of distinguishing accuracy from one candidate to another. It has been given a weight of 1.

The baseline's format of a 48 bit extended precision has been given a rating of 10 because of its accuracy.

The APU format is a unique 32 bit format and can be used in airborne applications. It has been given a rating of 6 because it does not currently conform to any of the standard or proposed standard formats.

The 2903 Add-On is a 32 bit user configurable format. It would be implemented to perform the IEEE format or the 1750A format via microcode. The 2903 Add-On has been given a rating of 8.

The Zilog EPU executes IEEE format, which is considered desirable and has been given a rating of 8.

The bit slice emulation is a 32 bit user configurable format, which can conform either with the IEEE standard or the 1750A standard. This is desirable and has been given a rating of 8.

The Intel combination executes IEEE format and is also considered desirable. The rating of 8 has been assigned to this candidate.

Table 18 shows the matrix of all candidates and their scores with respect to their criteria as well as the total rating of each candidate. The baseline has scored a 270 which, when compared to the APU and 2903 Add-On, is considered a better implementation. However, its high grade is mainly because of its compatibility with current software, but its execution is the slowest and could not be considered as the best candidate.

The APU has scored a 241 rating, which is the lowest rating other than the Intel rating. Its execution speed is not fast enough to meet projected requirements and therefore is not desirable.

The 2903 Add-On, with a rating of 265, is somewhat better than the APU candidate. However, its execution speed is marginal when compared to the last three candidates. This candidate would probably not be a desirable option in the DIS computer.

The Zilog EPU and the total bit slice emulation show the highest ratings, 281 and 280, respectively. The Zilog EPU has been penalized because of its lack of availability. If available, it clearly is the best candidate for implementation into the DIS computer since software impact is marginal, it has the cleanest hardware implementation, and the execution time is projected as the fastest of all candidates. The bit slice emulation as compared to the EPU is approximately the same. It is rated high because its hardware exists and is documented per AMD. However the hardware costs and design tasks are significant when compared to the EPU. It is a good candidate in the event of absence of the EPU. The total bit slice emulation would also allow emulation of the 1750A instruction set, which may be desirable in some military applications.

Of all the candidates, including the baseline, the Intel 8086/8087 combination has scored the lowest. This is not to suggest that the Intel version of floating point execution is a poor implementation. In fact, its execution is considered adequate to meet airborne requirements and has been basically penalized the most because of its incompatibility with current DIS software and hardware. For new computer developments, the Intel machine warrants significant re-evaluation. Intel has suggested that it is now committed to a military market and is in the process of validating second sources, working on licensing agreements with the military, and has committed to development of JOVIAL and ADA compilers.

RESULTS

The processing power of the Zilog Z8002 CPU can be boosted by an intrinsic capability of its bus called the extended processing architecture (EPA). This EPA allows the Z8002 CPU to accommodate up to four extended processing units (EPUs), which can perform specialized functions in parallel with the CPU's main instruction execution string.

Extended processors or parallel processors have been proven with large mainframe computers and minicomputers. They have shown that throughput is greatly increased by using this technique. In these systems, specialized functions such as array processing, trigonometric routines, or special I/O routines are typically assigned to extended processor hardware. In the DIS computer application, one extended processing unit would be dedicated to the execution of floating point arithmetic. The Zilog EPU is designed to perform its task on data resident in its internal registers. Moving information into and out of the EPU internal registers, as well as instructions, is the responsibility of the CPU.

The EPU connects directly to the Z8000 bus or Z bus and continuously monitors the CPU instruction stream (see Figure 46). The Z8002 normally fetches instructions, calculates the addresses of operands residing in memory, and controls the movement of data to and from memory. Because of the EPA, the EPU will monitor the activity on the Z bus. If the instructions fetched by the Z8002 are extended instructions, the EPU and the Z8002, latch the instruction. An ID field in the extended instruction will select a particular EPU (if more than one EPU is on the bus), then both the Z8002 and the indicated EPU will proceed to process the instruction. If the extended instruction indicates a transfer of data between an EPU's internal register and the main memory, then the Z8002 CPU calculates the memory address and generates the appropriate timing signals (address strobe, data strobe, etc.) allowing the data transfer to take place using the address lines. If a transfer of data between the Z8002 and the EPU is indicated by the extended instruction, the transfer occurs in a similar fashion under CPU control. If the extended instruction indicates an internal operation is to be performed by the EPU, the EPU begins execution of the task and the Z8002 will be freed to continue on to the next instruction in the program memory. The EPU processing proceeds simultaneously with the Z8002 processing until a second extended instruction is encountered by the CPU that is directed to the same EPU (if again more than one EPU is in the system) or upon EPU interrupt of the CPU. If the EPU is still executing the last extended instruction when the second extended instruction is encountered, the EPU temporarily suspends instruction fetching by the Z8002 through the use of the STOP line on the Z8002. This indicates to the Z8002 that the EPU is still executing the previous extended instruction. Figures 47 and 48 show the relationship between the CPU and EPU instruction flow.

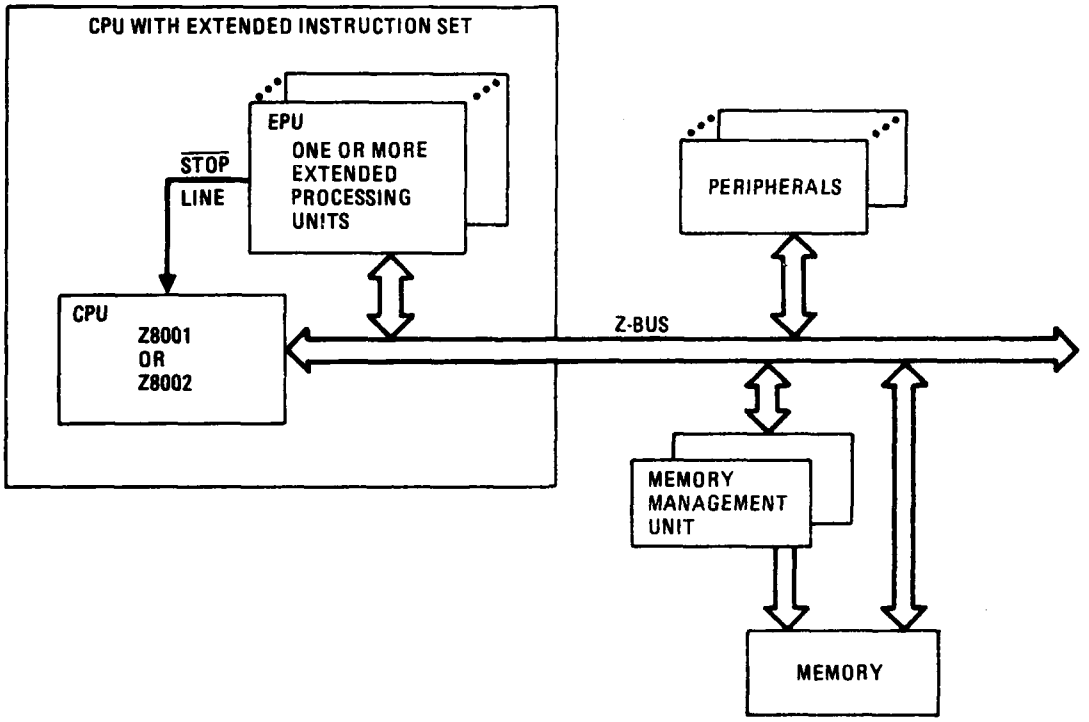


Figure 46. Typical EPU Configuration

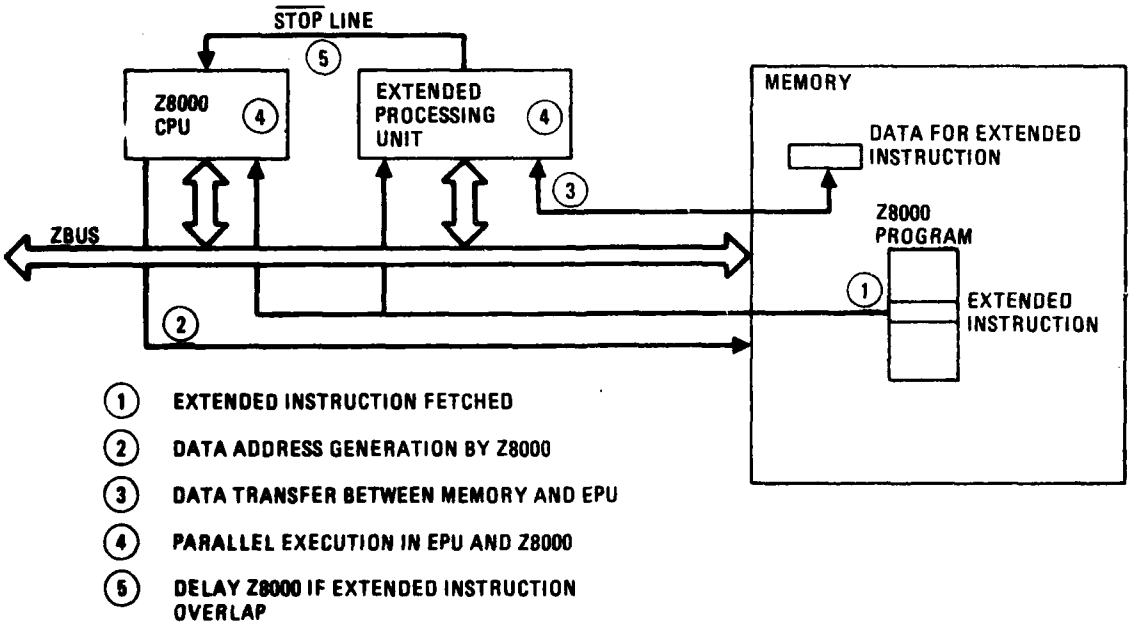


Figure 47. Execution of a Typical EPU Instruction

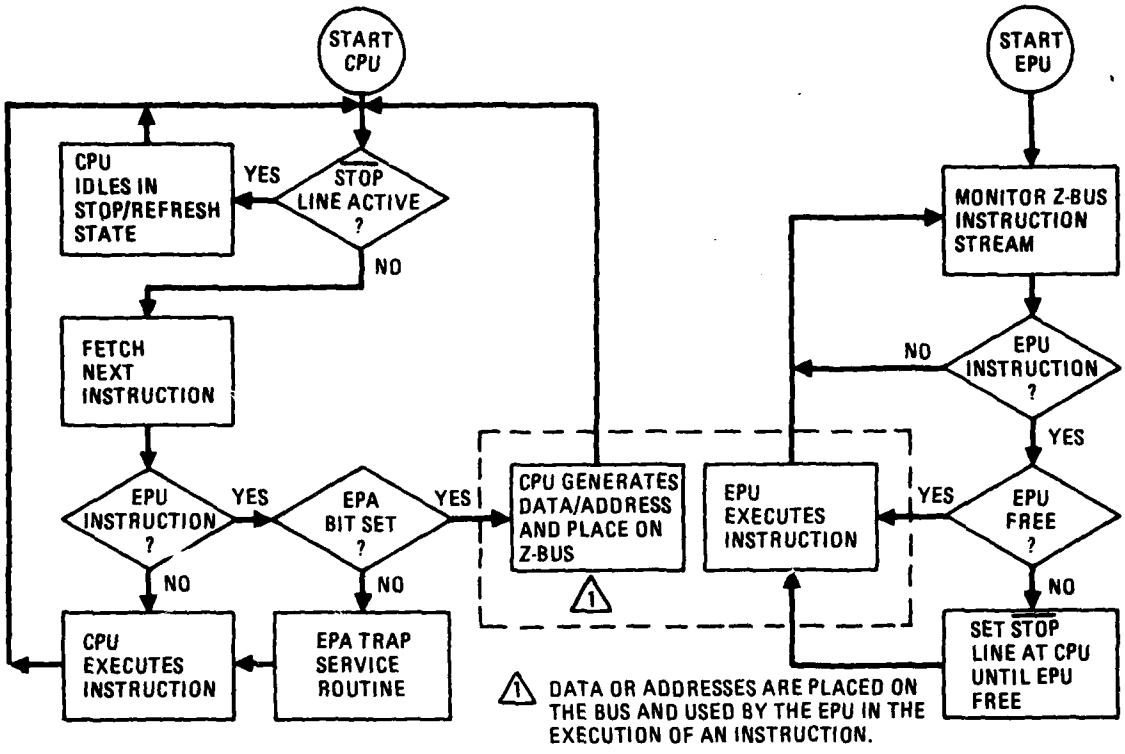


Figure 48. Flow Diagram of Z8002/EPU Execution Instruction

There are four classes of EPU instructions:

a. Data transfers between main memory and EPU registers.

b. Data transfers between CPU registers and EPU registers.

c. EPU internal operations.

d. Status transfers between the EPUs and the Z8002 CPU flag and control word register.

This last type of instruction is useful when the program must branch based on the results or conditions determined by the EPU. In addition, there are six Z8002 operation codes that are reserved for extended instructions. The action taken by the Z8002 upon encountering these instructions depends on a control bit in the Z8002 flag and control word registers. When this bit is set, it indicates that the system includes EPU(s) and the extended instructions should be processed as described previously. If the bit is not set, Z8000 traps so that a trap handler can emulate the desired EPU function in software if that software package is available to the system. This software package supports the debugging of suspect hardware against proven software. The trap mechanism

facilitates the design of systems for later addition of the EPU. (The software package for execution under a trap handler will be available July 1981. The EPUs from Zilog are available in a 1982-1983 timeframe.) It is possible therefore to write and execute software, verifying its accuracy, executing the extended instructions in non-real time, and placing the actual EPU in the circuit when available with no change to software. This is a significant advantage if this option is adapted early because all software changes to the operating system, the compiler, and task software can be written and verified immediately even in the absence of the EPU chip.

Figure 49 is an excerpt from the Zilog Z8002 technical manual. Condition codes or status codes of CPU and EPU interactions are described. Figure 50 shows an excerpt from the same Zilog document giving a detailed description of the EPU instructions.

The actual Zilog EPU chip is projected to be a 48-pin package with a slightly higher power dissipation over the Z8001. The pin out of the EPU will be similar to the Z8001 in order to ensure parallel access of the Z-bus. It will require a V-step CPU for proper operation (V-step CPUs are available). There will be somewhere between 8 and 16 internal registers, each capable of holding a triple precision number (96 bits). The Zilog EPU will execute floating point arithmetic in the sub "10 microsecond" timeframe and will use the standard IEEE format.

The Zilog EPU implementation clearly offers a simple interface to the existing DIS computer design with either the Z8002 or the Z8001 CPU since no other circuitry is required. However, the actual EPU product is not presently available and may not be available for immediate program needs. The EPU could be incorporated into a near term design by using Zilog EPU software emulation package to develop and verify software for a Z8002/EPU system in the absence of the actual EPU chip. In addition, all software requirements including changes and modifications to the JOVIAL compiler, the operating system, and the cross assembler could be done ahead of schedule. Acceptance testing and proofing of software algorithms could be verified with little or no program schedule impact.

The DIS operating system will require a modification of its data base and context switching logic to save and restore the EPU registers when task state changes occur.

The cross compiler would require a complete redesign of the current floating point. This would require a change in the current single precision format, inclusion of the EPU double precision format, and a redesign of floating point-related code generation from calls to run-time floating point to in-line generation of EPU instructions.

The cross assembler would require implementation of the EPU instruction set, testing of this implementation, and updating the assembler documentation.

A Z8000 CPU and one or more Extended Processing Units (EPUs) work together like a single CPU component, with the CPU providing address, status and timing signals and the EPU supplying and capturing data. The EPU monitors the status and timing signals output by the CPU so that it will know when to participate in a memory or EPU transfer transaction. When the EPU is to participate in a memory transaction, the CPU puts its AD lines in 3-state while $\overline{DS}$ is Low, so that the EPU may use them.

In order to know which transaction it is to participate in, the EPU must track the following sequence of events:

- When the CPU fetches the first word of an instruction (ST_3-ST_0 = 1101), the EPU must also capture the instruction returned by memory. If the instruction is an extended instruction, it will have an ID field which indicates (along with the second instruction) whether or not the EPU is to execute the instruction.
- If the instruction is to be executed by the EPU, the next non-refresh transaction by the CPU will fetch the second word on the instruction (ST_3-ST_0 = 1100). The EPU must also capture this word.
- If the first word of the instruction indicates the immediate addressing mode, the next one to 16 non-refresh transactions by the CPU will fetch the immediate data (ST_3-ST_0 = 1100, R/$\overline{W}$ = High, B/$\overline{W}$ = Low) one word at a time.
- If the instruction involves a read or write to memory, there will be zero or more program fetches by the CPU (ST_3-ST_0 = 1100) to obtain the address portion of the extended instruction. The next one to 16 non-refresh transactions by the CPU will transfer data between memory and the EPU (ST_3-ST_0 = 1000, 1001, 1010, or 1011). The EPU must supply the data (Write, R/$\overline{W}$ Low) or capture the data (Read, R/$\overline{W}$ High) for each transaction, just as if it were part of the CPU. In both cases, the CPU will 3-state its AD lines while data is being transferred ($\overline{DS}$ Low). EPU memory transfers are always word-oriented (B/$\overline{W}$ Low).
- If the instruction involves a transfer between the CPU and EPU, the next one to 16 non-refresh transactions by the CPU will transfer data between the EPU and CPU (ST_3-ST_0 = 1110).

Note that in order to follow this sequence, an EPU will have to monitor the $\overline{BUSACK}$ line to verify that the transaction it is monitoring on the bus was generated by the CPU. It should also be noted that in a multiple EPU system, there is no indication on the bus as to which EPU is cooperating with the CPU at any given time. This must be determined by the EPUs from the extended instructions they capture.

A final aspect of CPU-EPU interaction is the use of the CPU's $\overline{STOP}$ pin. When an EPU begins to execute an extended instruction, the CPU can continue fetching and executing instructions. If the CPU fetches another extended instruction before the first one has completed execution, the EPU must activate the CPU's $\overline{STOP}$ pin to stop the CPU (as described in Section 9.7) until the instruction completes execution.

Besides determining whether or not to participate in the execution of an EPA instruction, the EPU must determine from the first two instruction words

- Whether or not a memory access will be made and how many words of instruction will be fetched before the data is transferred.
- The number of words of data to be transferred for memory or EPU-CPU transfers.

Figure 49. Central Processing Unit and Extended Processing Unit Interaction

LOAD MEMORY FROM EPU

OPERATION: MEMORY ← EPU

THE CPU PERFORMS THE INDICATED ADDRESS CALCULATION AND GENERATES *n* EPU MEMORY WRITE TRANSACTIONS. THE n WORDS ARE SUPPLIED BY AN EPU AND ARE STORED IN n CONSECUTIVE MEMORY LOCATIONS STARTING WITH THE EFFECTIVE ADDRESS.

FLAGS/REGISTERS: NO FLAGS OR CPU REGISTERS ARE AFFECTED BY THIS INSTRUCTION.

EXECUTION TIME: 10 + 3n CYCLES.

CLOCK CYCLES

mode	dst	NS	SS	SL
0 0	IR (dst = 0)	11 + 3n	15 + 3n	18 + 3n
0 1	X (dst = 0)	15 + 3n	15 + 3n	17 + 3n
0 12	DA (dst = 0)	14 + 3n		

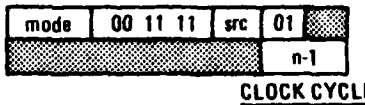
LOAD EPU FROM MEMORY

OPERATION: EPU ← MEMORY

THE CPU PERFORMS THE INDICATED ADDRESS CALCULATION AND GENERATES n EPU MEMORY READ TRANSACTIONS. THE n CONSECUTIVE WORDS ARE FETCHED FROM THE MEMORY LOCATIONS STARTING WITH THE EFFECTIVE ADDRESS. THE DATA IS READ BY AN EPU AND OPERATED UPON ACCORDING TO THE EPA *INSTRUCTION ENCODED INTO THE SHADED FIELDS.*

FLAGS/REGISTERS: NO FLAGS OR CPU REGISTERS ARE AFFECTED BY THIS INSTRUCTION.

EXECUTION TIME: 10 + 3n CYCLES



CLOCK CYCLES

mode	src	NS	SS	SL
0 0	IR (src = 0)	11 + 3n		
0 1	X (src = 0)	15 + 3n	15 + 3n	18 + 3n
0 1	DA (src = 0)	14 + 3n	15 + 3n	17 + 3n

LOAD EPU FROM CPU

OPERATION: EPU ← CPU REGISTERS

THE CONTENTS OF n WORDS ARE TRANSFERRED TO AN EPU FROM CONSECUTIVE CPU REGISTERS STARTING WITH REGISTER src. CPU REGISTERS ARE TRANSFERRED CONSECUTIVELY, WITH REGISTER 0 FOLLOWING REGISTER 15.

FLAGS/REGISTERS: NO FLAGS ARE AFFECTED BY THIS INSTRUCTION.

EXECUTION TIME: 11 + 3n CYCLES.

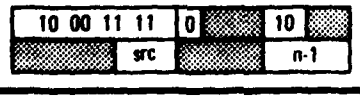


Figure 50. Extended Processing Unit Instructions

LOAD FCW FROM EPU

OPERATION: FLAGS ← EPU

THE FLAGS IN THE CPU'S FLAG AND CONTROL WORD ARE LOADED WITH INFORMATION FROM AN EPU ON AD LINES AD_0-AD_7.

FLAGS/REGISTERS: THE CONTENTS OF CPU REGISTER 0 ARE UNDEFINED AFTER THE EXECUTION OF THIS INSTRUCTION.

EXECUTION TIME: 14 CYCLES.

10 00 11 10		00	
	00 00		0000

LOAD CPU FROM EPU

OPERATION: CPU ← EPU REGISTERS

THE CONTENTS OF n WORDS ARE TRANSFERRED FROM AN EPU TO CONSECUTIVE CPU REGISTERS STARTING WITH REGISTER dst. CPU REGISTERS ARE TRANSFERRED CONSECUTIVELY, WITH REGISTER 0 FOLLOWING REGISTER 15.

FLAGS/REGISTERS: NO FLAGS ARE AFFECTED BY THIS INSTRUCTION.

EXECUTION TIME: 11 + 3n CYCLES.

10 00 11 11	0	00	
	dst		n-1

LOAD EPU FROM FCW

OPERATION: EPU ← FLAGS

THE FLAGS IN THE CPU'S FLAG AND CONTROL WORD ARE TRANSFERRED TO AN EPU ON AD LINES AD_0-AD_7.

FLAGS/REGISTERS: THE FLAGS IN THE FCW ARE UNAFFECTED BY THIS INSTRUCTION.

EXECUTION TIME: 14 CYCLES.

10 00 11 00		10	
	00 00		0000

INTERNAL EPU OPERATION

OPERATION: INTERNAL EPU OPERATION

THE CPU TREATS THIS TEMPLATE AS A NO OP. IT IS TYPICALLY USED TO INITIATE AN INTERNAL EPU OPERATION.

EXECUTION TIME: 14 CYCLES.

10 00 11 10		01	

Figure 50. Extended Processing Unit Instructions (Concluded)

CONCLUSIONS/RECOMMENDATIONS

The Zilog EPU has been clearly proven to be the best candidate for hardware fast floating point implementation into the current DIS computer. The hardware interface is the least complex, the software impact is minimal, and the execution speed of the EPU is the fastest. However, because of the lack of availability of the EPU, certain near term applications may not be met. It is possible to incorporate the EPU into the DIS computer during 1982 when the software emulation package from Zilog becomes available. This would allow total software coding at the assembly level and the high order level for floating point applications.

Because the software package will contain enough information so that work on the compiler and other software packages can proceed in 1981, the impact of no physical EPU chip may not be significant if the EPU chip is really available in the near future. Zilog states that it is committed to development of the chip, but because of limited human resources, the development will take time. They have stated that the EPU development is also a new type development when compared to existing Zilog product line. Therefore its development cycle is longer.

To maintain the legacy of the present DIS computer hardware/software, it is probably desirable to investigate the total bit slice emulation candidate. This candidate offers maximum use of the current DIS system both in hardware and software in the development of a hardware floating point, which would be fast enough to accommodate airborne applications. This candidate could be used as a stop gap for near term applications where fast floating point is required in lieu of the Zilog EPU.

The Intel 8086/8087 candidate is the only other candidate evaluated that offers a significant and acceptable floating point execution time. If costs and legacy of the current DIS computer were not taken into account, the Intel candidate would be a very viable solution. All Intel hardware and software does exist. Intel has committed to a JOVIAL compiler with plans of ADA support on their new machines. For a new computer application, the Intel candidate is probably the best where low risk is a major problem.

SECTION XIII

CONCLUSION/RECOMMENDATIONS

Specific conclusions concerning each of the aforementioned sections will not be reiterated here. However, this section does deal with the more global issues concerning this DIS contract as well as future considerations.

The total DIS computer system can be broken down into four major categories: airborne hardware, airborne software, software support, and test equipment. Included in each of these categories is a number of items that should be brought to the attention of future or follow-on efforts. These will now be discussed.

AIRBORNE HARDWARE

In the light of the delivered airborne DIS computer, a number of limitations should now be investigated. Class I and Class II machines may not offer enough selections for a very low-cost, low-throughput application as well as high-throughput applications. Efforts should be made to develop the lower end or Class 0 machine to provide the systems designer with a less flexible, but more cost effective computer for certain applications. This Class 0 machine is conceivably a single board computer with a few I/O cards and extremely small memory. In addition, improvements to our present interrupt structure (i.e., allowing interrupt stacking per I/O port and increasing the number of levels) should be investigated.

Power Supply

The extended family of airborne DIS computers would also need a flexible power supply, which could vary in capacity based on each respective computer's characteristics. Particular attention to system considerations in connecting these supplies (the grounding system) must also be included. The contract requirement for hot start, warm start, and cold start should probably be re-evaluated to determine if these requirements are necessary for a full scale engineering development (FSED) computer system or should they be altered. It may be advantageous in some future computer system to develop a power-fail power-start routine that would allow the system designer to "tuck away" valuable information in the event of a power failure and to recognize power-up and continue execution.

Memories

The delivered memories for the Class I and II computers were all RAM with battery backup. In the light of our recent qualification test problems in qualifying the Eagle-Pitcher battery, and considering the logistic problems of maintaining a battery in an FSED computer system, it is appropriate to investigate

newer memory technologies such as E^2 PROM to provide a "non-volatile RAM." Such a memory mechanism is currently being designed on programs such as Medium Range Air to Surface Missile (MRASM).

To complete the airborne hardware computer system, a mechanism for providing more than 64K words of memory is appropriate. Paging schemes could be used with the Z8002 to allow for more than 64K words of memory. These paging schemes would provide minimum impact on the present compiler and software packages. However, it may be more cost effective to pursue the Z8001 CPU, with or without the memory management unit (MMU), to provide for a more flexible hardware implementation to address up to 8 megabytes.

Potential users of this computer system may also have requirements for bulk storage which could include Block Organized RAM (BORAM), disk, cassette, and magnetic bubble memories. A means to implement any or all of these bulk storage mechanisms should be investigated and be provided for the system designer to use as he sees fit.

I/O

One of the objectives of the DIS contract was to establish standards in the I/O arena. Standardization with any of the identified I/O as delivered in the DIS computer seems impossible at this time. The DISMUX data bus, although technically attractive, has met with standardization opposition in many arenas. However, the merits (cost, implementation, latency, efficiency) of the DISMUX system cannot be overlooked.

The parallel handshake channel and the serial handshake channel as identified within the contract would serve little or no use in an actual system. Because they are programmed I/O, the CPU must spend enormous amounts of time servicing these boards (i.e., interrupt upon each word/byte transferred). The integration process with Lear-Siegler's guidance set (LCIGS) has demonstrated this fact. In this case, two cards (the SIO and PIO) had to be modified to make them appear to be DMA channels to the DIS computer. It is recommended that all I/O cards appear to be DMA channels in and out of these airborne processors.

The MGD program has demonstrated that a larger selection of I/O is required in an actual system application. A number of new I/O channels had to be designed and developed to support the DIS computer in the MGD application. Similar kinds of I/Os will probably have to be designed in other applications if this computer or future computers based on the DIS system are to be used.

Environmental Characteristics

Although the delivered computers under this contract have been tested for temperature, shock, vibration, etc., the levels and the extent to which they

have been tested should be recognized as a relatively non-extreme environment. It is therefore necessary for a future user of the DIS computer to increase the current capability of the delivered package to meet the environment of their particular application.

Some applications may require the use of a technology that will allow compatibility with a radiation environment. This is an outstanding problem for most military system houses. Few technologies (bipolar, CMOS/SOS) can meet the levels of radiation that would make them applicable in these kinds of weapons systems. Further work should continue in this area to provide a low-cost alternative to the present radiation hardened computers.

Packaging

The DIS system was successful in maintaining modularity within the airborne computer. This should be maintained, at least in a projected Class III machine, although it may not be necessary in a Class 0 machine. The modularity concept of mixing and matching I/O is desirable.

The delivered DIS computer packaging concept was driven by the requirement to maintain a volume under 150 cubic inches. Although a small package is highly desirable in many applications, it may be necessary to allow the package to expand to meet some of the thermal considerations and/or to provide militarized type connectors (i.e., MIL-C-38999). However, because of the ever increasing new technology and the newer IC packaging concepts being developed, a small size computer can still be provided even in the light of more stringent thermal considerations and the larger I/O connectors. An IC packaging concept that shows promise is the leadless chip carrier (LCC). Many manufacturers have already announced their products are available in leadless chip carriers and more will follow. Included in these are AMD, Zilog, Intel, Mostek, and TI. It is, therefore, possible to "shrink" the actual board area used in the implementation of various computer parts. In doing this, we could conceivably eliminate the use of "flat packs," IC packages that are costly to use in a production atmosphere, and may or may not be available in the future of the FSED program.

Manufacturing Considerations

If the DIS system is to be used in an FSED program (or a newer version of the system), manufacturing considerations must be investigated as to its feasibility, cost, etc. The DIS contract did not actually address these considerations as far as Quality Assurance, failure analysis, level of detail, etc. were concerned. An FSED program that considers using the DIS system still must decide what level of these parameters is appropriate.

AIRBORNE SOFTWARE

The delivered airborne software on the DIS contract included all task oriented modules written in JOVIAL J73 and an operating system written in assembly language. The advantages of high order language have been obvious to such programs as MGD in terms of ease of writing code. However, the efficiency of such code has yet to be realized. Additional work must be done to determine the efficiency, maintenance, use of optimizers, and resources necessary to use a high order language.

The operating system was developed by General Dynamics because there was no available OS for this development. However, industrial houses are now developing real-time operating systems that show pomise. Such a system is Zilog's new real-time operating system called Zilog Real Time System (ZRTS).

Future designs of the DIS system or similar systems must evaluate the requirement for a built-in test (BIT) capability. Such a capability in the strictest sense of the word is not available in the current delivered DIS system. To do so would require added memory plus added hardware, which may increase the size of the DIS computer. BIT can and should be added in an FSED computer development system. In addition to BIT, added diagnostics must also be developed for use at the computer level, board level, and ultimately, the vehicle level.

SUPPORT SOFTWARE

Compilers

The delivered JOVIAL compiler under this contract, although new and relatively immature, is a significant step in using a high order language in an airborne system. Maintenance and improvements need to be continued. However, other compiler efforts may be more appropriate. Such compiler developments obviously include ADA and possibly PASCAL or FORTRAN. Because of the size of these compilers, the computer host is a sizeable investment. The JOVIAL community appears to be basing all hosts on IBM machines or DEC machines. If this continues in other compiler developments the choice will at least be limited to these two families and therefore minimize the cost of computer maintenance. The decision to spend dollars on JOVIAL versus ADA is not clear cut.

Support Software Tools

Linkers, loaders, assemblers, and editors are all useful and necessary to the software programmer. Because of the proliferation of hosts for these tools, the maintenance of such tools could be extremely costly unless there was a "standard host."

Simulators

A desirable support tool for software that was not used on the execution of the DIS contract because of its lack of availability was a simulator. Current Z8002 simulators are not necessarily DIS simulators because of our delivered airborne computer. However, they still offer solutions to various software problems for the software programmer. As this tool develops, it should be investigated and utilized if appropriate.

Debuggers

The apparent weakness of the use of the JOVIAL language is a lack of a symbolic debugger or hex real-time debugger to support the language. This development is necessary to support an actual FSED program. To debug high order language code by the execution of such code in the actual target is very inefficient.

Software Development Philosophy

Much more work must be done to determine what is the best way to produce software, maintain software, and train personnel in software development of a particular system. Such questions have plagued computer houses as well as the military for years and will probably continue to do so. Because the cost of hardware is continuing to drop, and the cost of writing software to use that hardware is continuing to rise, this problem must be addressed. The standardization community, which believes that standardizing a single language will solve the problems, has started addressing this problem but it in itself will not solve the problem.

TEST EQUIPMENT

The DIS contractual test equipment is the delivered DDS system. This facility is insufficient to thoroughly check out the entire computer system or provide the cost effective software facility for a DIS-type system and FSED program.

Board Checkout

In a production atmosphere it is necessary to efficiently check out airborne hardware at the board level. The DDS system is not capable of doing this in a timely manner. Alternative type systems must be developed and utilized. A typical example is the Hewlett Packard DTS 70 test set.

Computer Level Checkout

In a production atmosphere, an efficient and thorough box or computer checkout system must be developed. The current DDS system, as is, does not answer this need but is certainly a start at this checkout. If software enhancements do take place on the current DDS system, a complete computer-level checkout

station could be developed; however, DDS costs seem to prohibit this. A low-cost version of the DDS could possibly meet the needs of this computer or box checkout station. For example, instead of using a PDP-11/34 as the basic machine, an 11/23 or 11/24 may be more cost effective and would still allow a capability to check computers in a production atmosphere.

Acceptance Test Environment

In a production atmosphere an efficient means of acceptance testing an airborne computer has yet to be addressed. The DIS contract did demonstrate the costliness and inefficiency related to checking out and acceptance testing each computer. In addition to thermal and vibration test fixtures, burn-in fixtures at the box level, board level, and possibly piece parts, a system test fixture should also be designed. The software to accomplish such an acceptance test is crucial in terms of being complete and maintainable.

Field Support Equipment and Depot Support Equipment

In an actual production system certain questions must be answered as to the applicability and availability of support equipment in the field. Should the field support equipment be the same as the factory equipment? Should a depot be allowed to go beyond vehicle checkout, go beyond box or computer checkout, go beyond board level checkout? Depending on the complexity of the weapon system and depending on the complexity of the computer system within that weapon, the answers to these questions will determine the equipment required. If the contractor continues to play an active role in field support and depot checkout, the military will continue to pay that added expense for the life cycle costs to maintain that weapon system.

FINAL COMMENTS

The execution of the DIS contract has shown a significant degree of success in providing an airborne, modular, distributed system that is programmable in a high order language. The real proof of the success of this contract will not be determined until follow-on contracts such as MGD continue the work developed here. Few computer houses today can offer the extent of computer development that can be offered with the DIS system. However, the completion of this computer system contract should not be the end of the road. Continued work is necessary to improve the concept and to make it applicable for low-cost tactical weapons systems.

BIBLIOGRAPHY

Computer Program Product Specification for the Digital Integrating Subsystem (DIS) Control, Monitor, and Load, Revision A, (CS64-32111), General Dynamics Convair Division, San Diego, CA, 17 April 1980.

Computer Program Product Specification for DIS Diagnostic Station Diagnostic Software, CS64-32112, General Dynamics Convair Division, San Diego, CA, 15 June 1981.

Computer Program Product Specification for the Digital Integrating Subsystem (DIS) Operating System, DS64-31301, General Dynamics Convair Division, San Diego, CA, 13 April 1980.

Computer Program Product Specification for the Digital Integrating Subsystem (DIS) Assembler, CS64-32201, General Dynamics Convair Division, San Diego, CA, 15 June 1981.

Computer Program Product Specification for the Digital Integrating Subsystem (DIS) Link/Loader, CS64-32202, General Dynamics Convair Division, San Diego, CA, 15 June 1981.

Computer Program Product Specification for the Digital Integrating Subsystem (DIS) Compiler, CS64-32203, General Dynamics Convair Division, San Diego, CA, 15 June 1981.

Critical Item Development Specification for the Digital Integrating Subsystem (DIS) Standard Interfaces, Revision A, CS64-31110, General Dynamics Convair Division, San Diego, CA, 21 August 1980.

Development Specification for the Digital Integrating Subsystem (DIS) for the Multiplex Bus (DISMUX), CS64-31200, General Dynamics Convair Division, San Diego, CA, August 1981.

Digital Processing Analyses/Partitioning (DPAP) Final Report, Draper Laboratory, Inc., November 1977.

DIS Diagnostic Station Computer Programming Manual, CS64-32221, General Dynamics Convair Division, San Diego, CA, 15 June 1981.

DIS Diagnostic Station Operation and Maintenance Manual, CS64-32200, General Dynamics Convair Division, San Diego, CA, 30 June 1981.

DIS Programming Manual, CS64-32222, General Dynamics Convair Division, San Diego, CA, 15 June 1981.

Fast Floating Point Trade Study, Digital Integrating Subsystem (DIS) Program, General Dynamics Convair Division, San Diego, CA, December 1980.

Federated Computer System Hardware/Software Development Engineering Report, GDC-ERR-79-053, General Dynamics Convair Division, San Diego, CA, 21 December 1979.

Low Cost Federated Computer Technology Development Engineering Report, GDC-ERR-80-023, General Dynamics Convair Division, San Diego, CA 23 December 1980.

Microcomputer Components Data Book, Zilog, Cupertino, CA, February 1980.

Military Standard MIL-STD-1553B Aircraft Internal Time Division Command/Response Multiplex Data Bus, Aeronautical Systems Division, Wright-Patterson Air Force Base, Ohio, September 1978.

Military Standard MIL-STD-1760 Draft, Aircraft/Store Electrical Interconnection System, Air Force Armament Laboratory, DLJA, Eglin AFB, Florida, February 1979.

Digital Integrating Subsystem (DIS) Diagnostic Programs User's Guide, CS64-32208, General Dynamics Convair Division, San Diego, CA, 28 September 1981.

Digital Integrating Subsystems (DIS) User's Guide for the Link/Loader, Revision A, CS64-32205, General Dynamics Convair Division, San Diego, CA, 30 January 1981.

Digital Integrating Subsystem (DIS) User's Guide for the Assembler, CS64-32206, General Dynamics Convair Division, San Diego, CA, 30 January 1981.

Digital Integrating Subsystems (DIS) User's Guide for Compiler, CS64-32210, General Dynamics Convair Division, San Diego, CA, 1 May 1981.

Digital Integrating Subsystem (DIS) User's Guide for the Control, Monitor, and Load Programs, CS64-32227, General Dynamics Convair Division, San Diego, CA, 30 June 1981.

Digital Integrating Subsystem (DIS) User's Guide for the Operating System CS64-32209, General Dynamics Convair Division, San Diego, CA, 12 May 1981.

Digital Integrating Subsystem (DIS) User's Guide for Arpanet, CS64-32230, General Dynamics Convair Division, San Diego, CA, 15 September 1981.

BIBLIOGRAPHY (CONCLUDED)

Digital Integrating Subsystem (DIS) Acceptance Test Plan/Procedures, CS64-30102, General Dynamics Convair Division, San Diego, CA 15 June 1981.

Prime Item Development Specification for the Digital Integrating Subsystem Diagnostic Station, Revision B, CS64-32100, General Dynamics Convair Division, San Diego, CA, 17 April 1980.

Z8000 CPU Technical Manual, Zilog, Cupertino, CA, August 1980.

LIST OF ABBREVIATIONS AND ACRONYMS

AD	Armament Development
AFAL	Air Force Avionics Laboratory
ALCM	Air Launched Cruise Missile
ALU	Arithmetic Logic Unit
AMD	Advanced Micro Devices
ARPANET	Advanced Research Project Agency Computer Network
ASCII	American Standard Code for Information Interchange
B/B	Buffer Box
BCD	Binary Coded Decimal
BIT	Built-In Test
BITE	Built-In Test Equipment
BIU	Bus Interface Unit
BOM	Beginning of Message
BORAM	Block Organized RAM
BR	Bus Request
CBLOK	Computer Block
CCA	Circuit Card Assembly
CMOS	Complementary Metal-Oxide Silicon
CML	Control, Monitor, and DIS Memory Load
C/O	Checkout
CPC	Computer Program Component
CPCI	Computer Program Configuration Item
CPU	Central Processing Unit
CRT	Cathode Ray Tube
CRX	Computer Receiver
CSER	Computer Serial
CTU	Control Terminal Unit
CTX	Computer Transmit
DAC	Digital-Analog Converter

LIST OF ABBREVIATONS AND ACRONYMS (CONTINUED)

DAP	Digital Autopilot
DCA	DIS Cross Assembler
DCASPRO	Defense Contract Administration Services Plant Representative Office
DDR	Digitally Derived Rate
DDS	DIS Diagnostic Station
DEC	Digital Equipment Corporation
DIP	Dual Inline Package
DIS	Digital Integrating Subsystem
DISMUX	DIS Multiplex Bus
DLL	DIS Linking Loader
DMA	Direct Memory Access
DMAC	Direct Memory Access Controller
DOF	Degree of Freedom
DRB	Design Requirements Bulletin
DTS	Digital Test Set
EMI	Electromagnetic Interference
EMP	Electromagnetic Pulse
EOT	End of Transmission
EPA	Extended Processing Architecture
EPROM	Erasable Programmable ROM
EPU	Extended Processing Unit
FIFO	First-In-First-Out
FDS	Federated Diagnostic Station
FORTRAN	Formula Translation (a programming language)
FPH	Floating Point Hardware
FSED	Full Scale Engineering Development
GAN	Guidance and Navigation
GD	General Dynamics
GDC	General Dynamics Convair Division

LIST OF ABBREVIATIONS AND ACRONYMS (CONTINUED)

GFE	Government Furnished Equipment
HOL	Higher Order Language
I^2	Intermessage Interval
IC	Integrated Circuit
ID	Identification
IIAS	Integrated Inertial Avionics System
ILS	Integrated Logistic Support
INR	Inertial Navigation Reference
INREQ	Input Request
INST	Instruction
INT	Interrupt
I/O	Input/Output
I/OC	Input/Output Controller
IP	Interface Processor
JOCIT	JOVIAL Compiler and Implementation Tool
JOVIAL	Jules Own Version of International Algol (a programming language)
JOVS	JOVIAL Compiler Verification Software
KOPS	Thousand Operations Per Second
LCC	Leadless Chip Carrier
LCIGS	Low Cost Inertial Guidance System
LSI	Large Scale Integration
LSTTL	Low-Power Shottky Transistor-Transistor Logic
ME	Message Error
MGD	Midcourse Guidance Demonstration
MMU	Memory Management Unit
MOS	Metal Oxide Silicon
MRASM	Medium Range Air to Surface Missile
MSB	Most Significant Bit
MSI	Medium Scale Integration

LIST OF ABBREVIATIONS AND ACRONYMS (CONTINUED)

MUX	Multiplexer
NPR	Non Processor Request
NRZ	Non Return to Zero
OS	Operating System
OT	Overtime
OUTREQ	Output Request
PAL	Programmable Array Logic
PBLOK	Peripheral Block
PCM	Pulse Code Modulation
PDP	Programmable Data Processor
PDSMC	Power Drop Save Memory Command
PHS	Parallel Handshake
PIOC	Parallel Input/Output Controller
PPAR	Peripheral Parallel
PROM	Programmable ROM
P/S	Parallel to Serial
PSE	Peculiar Support Equipment
PSER	Peripheral Serial
R&D	Research and Development
RADC	Rome Air Development Center
RAM	Random Access Memory
REQ	Request Lines
ROM	Read Only Memory
RRPP	Round Robin Passing Protocol
RTC	Real-Time Clock
RTU	Remote Terminal Unit
RTX-CLK	Receive-Transmit Clock
SES	Software Engineering System
SIL	Simulation Integration Laboratory
SIOC	Serial Input/Output Control

LIST OF ABBREVIATIONS AND ACRONYMS (CONCLUDED)

S/P	Serial to Parallel
SSDL	Software Design and Documentation Language
SSI	Small Scale Integration
STE	Special Test Equipment
SUP	Supervisor
TCB	Task Control Block
TCT	Task Control Table
TERCOM	Terrain Contour Matching
TTE	Transmission Time Out Error
TTL	Transistor-Transistor Logic
UV	Ultraviolet
UTG	Unaided Technical Guidance
VIP	Vehicle Interface Processor
ZRTS	Zilog Real Time System


DAT
FILM